Security and Optimization of Blockchains and Associated Algorithms

Soutenue par
Mirko KOSCINA
Le jj mois aaaa

École doctorale n°386
Sciences Mathématiques de Paris Centre

Spécialité
Spécialité Informatique

Résumé

La blockchain peut être définie comme un système distribué qui doit synchroniser létat actuel de la chaine ent re tous les nuds qui font partie du réseau. Ce processus de réplication de la blockchain est résolu de différentes manières selon le niveau danonymat utilisé par le système. Dans le cas de Bitcoin, la blockchain la plus connue, cest un réseau dit «*permissionless* »qui assure le complet anonymat de ses utilisateurs. Dans ce cadre, lalgorithme proposé par Nakamoto utilise un nouveau mécanisme de consensus pour faire la mise à jour de la blockchain basé sur un puzzle cryptographique dit «*Preuve de travail* »(«*Proof of Work* »). Plusieurs propriétés de la blockchain ont été développées et éprouvées comme le préfixe commun et la qualité de la chaîne dans un réseau synchrone, la vivacité et la consistance dans un réseau asynchrone ainsi que la prévention contre des attaques «sybil ». Cependant, il reste à mener un travail important de recherche sur la confidentialité de données pour dautres architectures de la blockachain, comme par exemple celle de : *permissioned.* En effet, la blockchain *permissioned* noffre pas lanonymat offert par le Bitcoin. Ainsi, il manque des mécanismes pour assurer la confidentialité de lidentité des utilisateurs. De plus, la technologie blockchain doit sadapter au nouveau règlement général européen sur la protection des données (RGPD) imposant un nouveau cadre pour la gouvernance des données, pour que les propriétés de cette technologie, comme limmuabilité, soient compatibles avec la nouvelle réglementation.

Bien que les défis liés à la confidentialité de l'utilisateur et à la gouvernance des données soient importants, les principaux cas d'usage de la blockchain sont les crypto-investissements et la spéculation sur les actifs numériques, et non les services impliquant des données personnelles. Même dans le domaine des crypto-monnaies, il n'y a pas de travaux de recherche étendus qui abordent les nouveaux problèmes économiques auxquels nous sommes confrontés, qui permettraient le développement de crypto-monnaies pour les économies locales ou circulaires.

Cette thèse aborde les problèmes de sécurité, de confidentialité et des barrières économiques auxquels fait face la mise en uvre de la blockchain pour assurer la confidentialité des utilisateurs, la gouvernance des données, l'offuscation des logiciels et développer des modèles de crypto-monnaies pour les économies circulaires. Après avoir introduit la blockchain et la cryptogra-

phie moderne, ils se présentent trois principaux résultats de recherche : un nouveau protocole de consensus qui préserve la confidentialité des utilisateurs, un nouveau schéma de vote électronique confidentiel utilisant une blockchain *permissioned* et un protocole doffuscation de graphiques de flux de contrôle pour la confidentialité des logiciels. Enfin, la troisième partie de la thèse comprend deux résultats de recherche appliquée avec deux prototypes de systèmes blockchain répondant aux problématiques posées par la gouvernance des données et les crypto-monnaies.

La première partie de cette thèse décrit un bref historique de la technologie autour de la blockchain et son évolution jusqu'à arriver à sa première implémentation (le Bitcoin). Ensuite, sont introduits les principaux sujets sur la cryptographie moderne utilisés dans cette thèse. La deuxième partie commence par un nouvel algorithme de consensus qui préserve la confidentialité pour la blockchain *permissioned.* La principale contribution de ce travail est une construction générale de signature aveugle pour la dissociation des transactions intégrée à un consensus basé sur le BFT pour la validation des transactions et des blocs. Le deuxième résultat correspond à un nouveau schéma de vote électronique pour la blockchain *permissioned.* Ce travail a propose un nouveau consensus de préservation de la confidentialité basé sur la signature aveugle d'Okamoto-Schnorr pour la dissociation des transactions et la courbe elliptique pour la confidentialité des votes. Enfin, il est présenté un nouveau mécanisme d'offuscation de graphes de flux de contrôle pour la confidentialité du source code des logiciels (par exemple, pour les contrats intelligents). La troisième partie présente deux implémentations de la blockchain. Le premier est une nouvelle implémentation de la norme de jeton ERC20 et une marketplace sur Hyperledger Fabric pour les économies circulaires supportées par le recyclage du plastique. Le second est un prototype de blockchain pour la gouvernance des données dans l'échange de données de santé selon le RGPD automatisé avec des contrats intelligents.

Abstract

Blockchain can be defined as a distributed system that synchronizes the current state into all the nodes that are part of the network. This replication process of the blockchain is solved differently according to the level of anonymity used by the system. In the case of Bitcoin, the best-known blockchain implementation, it is configured as a Permissionless network that assures the complete anonymity of its users. In this scenario, Nakamotos algorithm presents a novel consensus mechanism to update the blockchain based on a cryptographic puzzle or Proof of Work. Some properties of the Bitcoin protocol have been studied and proved, such as the common prefix and chain quality in a synchronous network, liveness and consistency in an asynchronous network, and prevention against Sybil attacks. However, research work on user and data privacy for other blockchain architecture, i.e. permissionless, has not been addressed comprehensively. Moreover, in Permissioned blockchain, the anonymity offered by Bitcoin does not exist. Hence, there is a lack of privacy-preserving mechanisms to ensure the identity privacy of the user. In addition, blockchain has been confronted with the new European General Data Protection Regulation (GDPR), which has imposed a new framework for data governance, making it very difficult to reconcile the new data regulation with blockchain's most important features, the immutability.

Although the challenges related to the user's confidentiality and data governance are significant, the most relevant use cases for blockchain are the crypto-investment and assets speculation, and not services that involve personal data. Nevertheless, even in the cryptocurrencies field, no comprehensive research work addresses the new economic problems we face, like cryptocurrencies for local or circular economies.

This thesis addresses the security, privacy, and economic barriers to implementing blockchain for user's privacy, data governance, software obfuscation and cryptocurrencies models for circular economies. The first part of this thesis covers the preliminaries about blockchain and modern cryptography. The second part presents three main results concerning privacy-preserving consensus, privacy in electronic voting with permissioned blockchain, and control-flow graph obfuscation for software confidentiality. Finally, the third part includes two results based on applied research work

with two prototypes of blockchain systems for data governance and cryptocurrencies.

The first part of this thesis describes a brief history of the technology around blockchain and its evolution until arriving at the first blockchain implementation (Bitcoin). Then, the most relevant topics on Modern Cryptography used in this thesis are introduced.

The second part begins with a new privacy-preserving consensus algorithm for permissioned blockchain. The main contribution of this work is a general construction of blind signature for transaction unlinkability integrated to a BFT-based consensus for transactions and blocks validation. The second result corresponds to a new electronic voting scheme for permissioned blockchain. This work proposed a new privacy-preserving consensus based on Okamoto-Schnorr blind signature for transaction unlinkability and elliptic curve for vote privacy. Finally, a new control-flow graph obfuscation mechanism for software confidentially (e.i for smart contracts) is presented.

The third part presents two blockchain implementations. The first is a novel implementation of the ERC20 token standard and a Marketplace on Hyperledger Fabric for circular economies based on plastic recycling. The second one is a blockchain prototype for data governance in healthcare data exchange according to GDPR automatised using smart contracts.

Acknowledgement

I would like to thank David Naccache for opening the doors to my PhD. Without his guidance, support and opportunities that he has given me, this thesis would not have been possible. I would also want to thank Pascal Lafourcade for the guidance, encouragement, and scientific and personal support during the thesis.

I express my recognition to Moti Young, Olivier Blazy, Marine Minier, Jean-Guillaume Dumas, Melek Önen and Pascal Lafourcade for agreeing to serve in this thesis committee. I express my particular gratitude to my thesis referees Olivier Blazy and Marine Minier, for your availability and dedication.

Thank you sincerely to Natacha Laniado. You have been part of this journey since the first day. Without your dedication, attention, and emotional support, especially during the most difficult moments, it would not have been possible to finish this thesis.

Finally, I want to thank my life partner Marion Jaffré. I thank the the universe for our paths crossing. Your love, support, encouragement, and patience have been the fuel to help me work and finish this thesis. There are no words in any language to express my love gratitude to you.

Contents

List of Figures

Part I

Prologue

Chapter 1

Introduction

Contents

Blockchain technology can be defined as a distributed database that is based on records organized as a chain of blocks. A peer-to-peer network performs the management, updates, operation, and governance of this distributed database. One of the main characteristics of blockchain is its resistance to malicious modifications. This high-security property is reached by using block timestamp and hash pointers that link the last block of the chain to the previous one and a strong mechanism to prevent malicious peers from adding new blocks to the chain. The blockchain's design implies that any modification made to a block forces the regeneration of the next blocks attached to the chain. This exhaustive process is extremely difficult to achieve. The addition of new blocks in the chain corresponds to the replication of system world state and ensures that the blockchain is continuously updated. This process is called consensus, and a large number of algorithms based on different cryptographic challenges or economic penalties are currently available. Consensus provides the certainty that only an honest node will update the main chain. The process for selecting the honest node that has the right to add a new block will depend on the type of blockchain implementation. The most popular consensus technique in the blockchain is Proof-of-Work, which solves a cryptographic puzzle [Nak08]. Other consensus schemes are based on the number of their assets, such as Proof-of-Stake [KN12] or the agreement between the network peers in a democratic scheme such as PBFT [CL99].

This chapter will cover a brief introduction of blockchain, starting from its birth and its evolution until today, to then describe the main characteristics and architecture of the permissionless (public) and permissioned (private) blockchain.

1.1 The Birth of Blockchain

Distributed systems have been studied in the research field since long before the blockchain arrival. For example, the idea of chaining blocks with a cryptographic hash function for data immutability was proposed by Ralph Merkle in his PhD dissertation, in 1978 [Mer79]. Merkle showed in this work that the information could be linked efficently organising the data in a tree structure. In the same year, David Chaum proposed that it is possible to create a trustful system based on a network of peers who do not trust one another [Cha79]. The building blocks of this system include physically secure "vaults," existing cryptographic primitives (symmetric, asymmetric and hash cryptographic functions), and a new primitive called Chaum-threshold secret sharing. In this system, each vault is responsible for signing, recording and broadcasting all the transaction processed by each one.

By the end of the 80s, Dwork, Lynch and Stockmeyer had shown the capacity of solving consensus in a broad family of "partially synchronous" systems [DLS88] in other to allow that a distributed system agreed on a result or message. Then, in 1990 Haber and Stornetta proposed a practical solution for time-stamping digital documents so that they could not be backdated, modified or damaged [HS90]. The system used a cryptographic chain of blocks to store the time-stamped documents. Then, in 1992 they added to the design the Merkle tree construction, making it more efficient by enabling many documents to be gathered into one block [BHS93a]. Later, In 1999, Castro and Liskov have proposed an efficient new data replication algorithm capable of resisting Byzantine faults in an asynchronous environment [CL99].

In late 2008 a white paper called Bitcoin: A Peer-to-Peer Electronic Cash System [Nak08] was published in a cryptography mailing list by a person or group of users using the pseudonym Satoshi Nakamoto. This white paper described an electronic cash system that allows online payments between peers without needing a central authority for the settlement process. The white paper argues that the system is capable of solving double-spending by using the peer-to-peer network protocol. The network timestamps each transaction by hashing and keeping them in a chain of blocks generated by using a Proof-of-Work (PoW) algorithm. This PoW is based on the Hashcash proof-of-work algorithm proposed in 1997 but formalised in a paper in 2002 [B$^+$02]. The idea of this Proof-of-Work is that one individual peer connected to the network can propose a new block with the transactions, and the rest of them are responsible for validating it. Thus, the first peer that calculates the Proof-of-Work is the one who wins the capacity to propose a new block and be rewarded for this work.

On the 3rd of January 2009, the Bitcoin came alive after the first mined block was appended to the chain. Satoshi Nakamoto mined the first block, and he was rewarded with 50 bitcoins. The first Bitcoin recipient was Hal Finney, who received ten bitcoins from Satoshi Nakamoto in the world's first bitcoin transaction on the 12th of January 2009.

In 2013, programmer and co-founder of the bitcoin magazine Vitalik Buterin exposed the bitcoin community with the need for a scripting language to build applications running on top of a blockchain network. After this, he started with the Ethereum project for building a new distributed computing system based on blockchain with the scripting functionality to create distributed applications, called smart contracts. In the beginning, the smart contracts were distributed programs or scripts that were deployed and executed on the Ethereum blockchain; they can be used to make a transaction if certain conditions are met. Smart contracts are written in specific programming languages and compiled into bytecode. These applications are executed on top of the Ethereum Virtual Machine (EVM), a decentralized Turing-complete virtual machine [Woo14]. Nowadays, we can see the use of smart contracts in different blockchain platforms.

The main principle of the EVM is to allow developers to create and publish applications that run inside Ethereum blockchain. These applications are also called decentralized applications or DApps. Nowadays, there are more than hundreds of DApps running in the Ethereum blockchain, including social media platforms, gambling applications, and financial exchanges. Additionally, Ethereum has its native cryptocurrency (like bitcoin) called Ether. This cryptocurrency can be transferred between accounts and is used to pay the fees for the computational power used when executing smart contracts.

The history of blockchain is recent; however, the blockchain backbones have been proposed long before Satoshi Nakamoto has published the Bitcoin White paper. The Table 1.1 presents a timeline of the evolution of some cryptographic discoveries until the first blockchains systems have disrupted our technological ecosystem.

Today blockchain technology is gaining a lot of mainstream attention and is already used in various applications, not limited to cryptocurrencies.

Table 1.1 – Timeline of the technological evolution of blockchain

1978	Merkle, Cryptographic puzzles [Mer79].
1979	Chaum, Vaults and secret sharing [Cha79].
1988	Dwork and Stockmeyer, Soliving consensus [DLS88].
1990	Haber and Stornetta , Time-stamping digital documents [HS90].
1999	Castro and Liskov, PBFT [CL99].
2002	Back, Hashcash [B^{+}02].
2008	Nakamoto, Bitcoin [Nak08].
2015	Buterin, Ethereum[Woo14].

Additionally, today there are several blockchain platforms based on different architectures (permissionless and permissioned). The following sections of this chapter cover the fundamental aspect of blockchain.

1.2 Blockchain Fundamentals

Blockchain is a secure and distributed ledger that sorts the transactions records hierarchically into a growing chain of blocks. The blocks keep the integrity of the records based on cryptographic hashes that link the list of transactions (information) with the blocks appended to the same chain. In addition to information about transaction records and their respective hash value, each block will keep a hash value of the block itself and the hash of the previous block (See Figure 1.1). This model allows maintaining a cryptographic fingerprint of the block (hash of the block) and a cryptographic pointer to the previous block (hash of the last block appended to the chain).

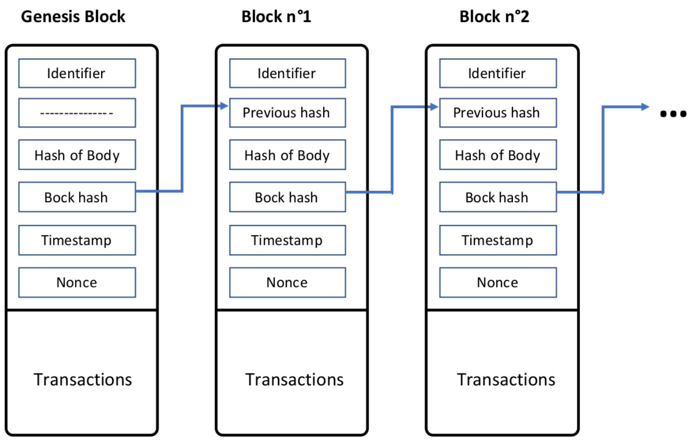


Figure 1.1 – Blockchain structure

Each peer will commit the new blocks upon successful competition of the decentralized consensus procedure in the blockchain. The consensus algorithm can vary depending on the blockchain architecture. However, their main goal is always to enforce the blockchain platform's security rules and business logic. The main controls that the consensus enforce are the proposal of new blocks into the blockchain, the read protocol for secure verification of the blockchain, and finally, the consistency of the data records included in each local version of the blockchain kept by each node.

As a result, the blockchain ensures that once a transaction record is added inside of a block. The transaction cannot be modified or tampered with; after this block has been successfully proposed, validated, and committed into the chain. Thus, the data consistency in each block of the blockchain is guaranteed, and the blocks, once committed into the blockchain, cannot be easily modified. Hence, blockchain technology proposes a suitable solution for a secure and distributed ledger, which archives all transactions between any two parties of an open networked system effectively, persistently, and in a verifiable manner.

Finally, the blockchain is a network that combines, in an elegant way, different cryptographic components like hashes and signatures, techniques from distributed systems like the consensus algorithm, and in some of them, a new layer of abstraction that allows running distributed programs called smart contracts.

1.2.1 The hashes

The hashes are a fundamental brick of blockchain technology. There are used to prove the integrity of the blocks and create a link between consecutive blocks that built the chain. Hence, it is possible to exemplify the use of the hashes as the digital fingerprint of each ledger page (the block) and the glue that assembly each ledger page (the chain).

The hash pointers: are used for chaining consecutive block. They are used to points to the location in which the data is stored (see Figure 1.1). Therefore, a hash pointer can be used to check whether or not the data has been tampered with. The chain of blocks is created by pointing to the predecessor block. The hash pointer indicates the address where the data of the predecessor block is stored. Additionally, the hash of the stored data can be publicly verified by users to prove that the stored data has not been modified.

If an adversary tries to modify the data in any block belonging to the chain, the adversary must change the hash pointers of the previous blocks to hide the tampering. Therefore, the adversary needs to change all pointers just to the first block (genesis block), which is generated once the system has been created. This implies that the adversary needs to invest a lot of resources to succeed in his attack. Finally, the hash pointers force the adversary to rebuild the chain to succeed in his attack; otherwise, the tampering will be uncovered.

The Merkle Tree: is a hash tree corresponding to a binary search tree with its tree nodes linked to one another using hash pointers. Every leaf node is tagged with a hash of a data block in the tree, and every non-leaf node is tagged with the hash of the tags of its child nodes (see Figure 1.2).

One of the main features of the Merkle tree is to prevent data tampering by searching down through the hash pointers to any node in the tree. More precisely, when an adversary tries to modify the data at a leaf node, it will

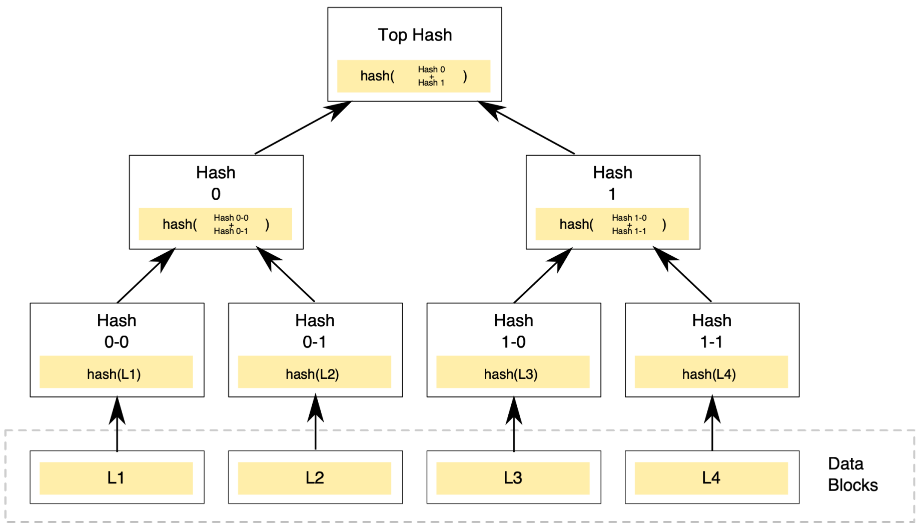


Figure 1.2 – The Merkle tree

produce a change in the hash value of its parent node. Hence, even if the adversary continues to modify the upper node, he needs to change all nodes on the path until the top. Using the Merkle tree, it is easy to detect if the data has been tampered with, since the hash pointer of the root node does not match with the hash pointer stored.

1.2.2 The permissionless and permissioned architecture

Blockchain technology has evolved rapidly in the last few years, and currently, it is possible to find different types of blockchain network architectures. These new architectures were born due to the different use cases and ecosystems where it is possible to implement services based on blockchain. The first official blockchain network was Bitcoin, which was designed to be fully open for users and peers. Hence, anyone in Bitcoin can be part of the network as an address owner for cryptocurrency trading or as a miner to be part of the peers that maintain the network updated and running. However, after some years, blockchain implementations in private companies turns the idea from the fully permissionless blockchain into a new kind of architecture called permissioned blockchain. This network architecture keeps the decentralization configuration of the system; however, it restricts access to the services running on the blockchain; for example, to be part of the consensus or to read and write in the blockchain.

The permissionless architecture allows us to have public blockchains;

however, in permissioned architecture, we can build different configurations according to specific needs. In this case, we can configure a blockchain network with all the nodes fully controlled by one organisation. This configuration is called a private blockchain. On the other hand, it is also possible to configure a blockchain where the nodes belong to different organisations or stakeholders. This configuration is called consortium blockchain. Hence, the main three blockchain network architecture identified are categorized as follow:

Permissionless blockchain: is a public blockchain implementation that anyone can query, write and be part of the mining. This scheme lets anyone with an address in the network (there is no identity linked to the address) execute transactions. It is also open to anyone that wants to be part of the blockchain processing services by running consensus protocol with the proper hardware or using their own stakes. This openness implies that the written content is readable by any peer. Nowadays, the most widely used permissionless blockchains are Bitcoin and Ethereum.

Private blockchain: is permissioned blockchain implementation where each network user must be enrolled in a central authority before joining it. Additionally, in this configuration, the nodes are controlled by one organisation. Hence the maintenance and running of the blockchain network depend on one entity. Read permissions may be public or restricted to an arbitrary extent. Likely applications include database management, auditing, etc., internal to a single company, and so public readability may not be necessary in many cases. Even though, in other cases, public auditability is desired. The most widely known instance of permissioned blockchains is Hyperledger Fabric.

Consortium blockchain: is also known as public permissioned blockchain. This ledger is a hybrid implementation between a Permissionless and a Private ledger. At the consensus level, the nodes responsible for maintaining the blockchain are pre-selected and belong to different organisations. At the users level, it may be fully open or private (i.e. users may be enrolled through an administrator or CA). This will depend on the type of services implemented in the blockchain. The specific configuration of each consortium chain; for example: which nodes can authorize transactions via the consensus process, which can review the history of the chain, which can create new transactions, and among others, it is a decision of each individual consortium.

1.2.3 Smart contracts

A smart contract can be defined as a computer program designed to execute automatically, control or document events according to the terms of the contract or the business logic implemented on it. Although this definition applies to smart contracts in several contexts, there is no single definition. For example, Roger Wattenhofer defines the smart contract as an agreement between two or more parties, encoded so that the correct execution is guaranteed by the blockchain [Wat17]. In Ethereum, the smart contracts are defined as [Sol18]:

> A contract [...] is a collection of code (its functions) and data (its state) that resides at a specific address on the Ethereum blockchain

According to Hyperledger Fabric [Fab20] :

> A smart contract can, for example, be written to stipulate the cost of shipping an item that changes depending on when it arrives. With the terms agreed to by both parties and written to the ledger, the appropriate funds change hands automatically when the item is received

Although there are several definitions of smart contracts, in this thesis, the smart contract is defined as decentralized applications (Dapp) that allow developers to create programs that run on a blockchain and use the computing power of the nodes connected to the blockchain network.

1.2.4 Consistency and liveness

There are several properties to consider in blockchain technology depending on the application and the network's architecture. However, there are two main properties transverse to any blockchain platform to ensure the proper functioning of the distributed network. In this thesis, those properties are called Consistency and Liveness.

Consistency: The consistency definition is not unique. Some authors define consistency as the property that ensures that all the nodes have exactly the same ledger at the same time. In this thesis, the definition of consistency is related to the capacity to ensure that the records stay immutable meanwhile the blockchain is growing. This means that the blockchain must ensure that once the peers have committed a block, it cannot be modified after a minimum number of blocks have been added to the chain. As it was explained before, one of the mechanisms to avoid malicious modification in a block is the point hashes, meaning that if an adversary wants to alter data in the last block appended to the chain, he does not need to modify the whole chain. However, if the adversary wants to modify a block generated several rounds before, he will be forced to rebuild the chain after the block he tampered with to hide the attack. Therefore, it is possible to define the consistency of the data stored in the ledger according to the amount of block added to the chain afterwards.

In this thesis, consistency is defined as follows:

Definition 1.1. *A protocol $\mathbb{P}$ is $T-$consistent if a transaction tx generated by an honest client c_{cb} to execute a valid operation o_{bc}, it is confirmed and stays immutable in the blockchain after $T-$ round of new blocks.*

Liveness: The liveness property is crucial in the blockchain. One of the main advantages of any distributed system is keeping the uptime or the service availability close to 100%. Hence, in this thesis, liveness means that a consensus protocol ensures that if an honest user submits a valid transaction, a new block will be appended to the chain with the transaction in it. Hence, the protocol must ensure that the blockchain grows if valid users generate valid transactions.

In this thesis, consistency is defined as follows:

Definition 1.2 (Liveness)**.** *A consensus protocol $\mathbb{P}$ ensures* liveness for a blockchain *C if $\mathbb{P}$ ensures that after a period of time t, the new version of the blockchain C' is $C' > C$, if a valid client $c_{i_{bc}}$ has broadcasted a valid transaction tx_i during the time t.*

1.3 Consensus Protocols

1.3.1 Proof-of-Work

The most popular Proof of Work protocol used nowadays is Nakamotos consensus, which is the backbone of Bitcoin. This algorithm is widely used in permissionless blockchain. In this architecture, the miners are anonymous; hence, there is no control over the identity and the access of network peers. Therefore, democratic mechanisms for consensus like the Byzantine Agreement are not suitable because an attacker can manipulate the system using a Sybil attack. This attack consists of spawns multiple peers, controlled by the attacker, and getting control of the majority of the network to manipulate the consensus result. The Proof-of-Work overcame this issue by forcing each peer to win the right to propose a new block showing that they are honest by winning a cryptographic challenge. Then the winner of the challenge is rewarded for the services rendered to the blockchain network.

The protocol consists of each participant having its local version of the blockchain. Each block consists of $(h_{-1}||\mu||m)$, where $||$ is the concatenation operator, h_{-1} is a hash pointer to the previous block in the chain, m are the records of the blocks, and μ is the solution of the cryptographic challenge (a.k.a. Proof of Work) derived from the hash of the previous block and the records. The protocol is parametrized by a parameter p, which represents the hardness of the challenge. The Proof of Work is solved once $H(h_{-1}||\eta||m) < D_p$, where $H(\cdot)$ is a hash function, η is a value selected by the miner (nonce), and D_p is the difficulty of the puzzle set in order to the probability that an input satisfies this relationship is less than p:

$$\Pr(H(h||\eta||b) < D_p) \leq p$$

According to Nakamoto's Proof of Work, the world state replication process proposes a new block to be committed by each node. Let's say that a block $b = (h_{-1}, \eta, m, h)$ is valid with respect to their predecessor block $b_{-1} = (h_{-1}, \mu', m', h')$ if only if:

- $h_{-1} = h'$
- $h = H(h_{-1}||\eta||m)$
- $h < D_p$

Additionally, the chain $(b_0, \ldots, b_j)$ is valid (block sequence states) if:

- $b_0 = (0, 0, \perp, H(0, 0, \perp))$ is the genesis block
- For all $i \in [j]$, b_i is valid respect to b_{i-1}
- The validity predicate $V(\cdot)$ of the records chain $V(C(state)) = 1$

Hence, each replication round proceeds as follow:

- All the incoming messages are read. If an incoming message *state*' is a valid sequence of blocks that is longer than its local version state, replace state by *state'*
- Read the record m (transactions). If m is such that $V(C(state)||m) \neq 1$, proceed to the next block round. Otherwise, select a random η and send a query $h = H(h_{-1}||eta||m)$. If $h < D_p$ then add the newly mined block to state (local chain) and broadcast the updated *state*.

In the case of Bitcoin, the PoW is based on Hashcash [B+02], which uses exhaustive searching of a value that when is hashed (SHA-256) twice with the block contents and a nonce, the result begins with a certain number of zero bits (target). Then, the nonce is increased until obtaining result compliance with the target. This challenge requires an average work that increases exponentially with the numbers of zero bits necessary to reach the target. This target is adjusted according to the network's processing performance to maintain the average time of new blocks in about 10 minutes.

1.3.2 Proof-of-Stake

The Proof-of-Stake (PoS) is an alternative type of distributed consensus protocols. It changes the view of proving honesty to get rewarded for security by promoting economic penalties. Compared to PoW used in Bitcoin and Ethereum, any blockchain network participant can participate in the consensus protocol or mining to create new blocks by solving cryptographical puzzles. In Poof-of-Stake blockchains, the protocol set the constraint on who can be chosen as miners. In PoS, only those participants who have locked up their stakes can be selected as miners or validators.

In Proof-of-Stake, the selection process can be done by randomized block selection or coin age-based selection. The first alternative randomizes the process by a formula that uses the hash value (search the lowest value) and the size of the stake. The second option combines a random process and coin-age, which is a value calculated according to the number of coins and the length of time they have been held.

PPCoin [KN12] was one of the first blockchain networks with Proof of Stake as a consensus algorithm. This protocol defines a new type of transaction block called Coinstake. In this block, the owner consumes their coin age (uses his own stake) in other to gain the privilege to propose a new block. The first input of this new block is called Kernel. This input requires to meet a hash target value similar to the Proof-of-Work, but the searching process is done in a limited search space. The space in PPCoin is limited by one hash per unspent wallet output. The hash target value is set according to coin age spent in the Kernel. Thus, the more coin gain spent in the Kernel, the target value is easier to meet. Once the block generation process is done, this will be added to the main chain, corresponding to the one with the highest total consumed coin age.

Another more recent Proof-of-Stake protocol is the launch with Ethereum version 2 called Casper. The Casper Proof-of-Stake protocol corresponds to a penalty-based PoS protocol. Once a new block is proposed in this protocol, only the nodes that wish to add on to their blockchains local version will place a bet (a portion of their stake) on it and become a validator. The validators use a portion of their assets (ETHs) as stakes for participating in the validation of the next blocks. The validators are rewarded proportionally to their bets on the block only when the block is committed into the blockchain. If a validator acts maliciously, it will be punished by losing its stakes. In the Casper protocol, once a peer has a malicious behaviour, it has something to lose; hence, the consequences of its acts are high.

1.3.3 Byzantine Fault Tolerance (BFT)

The Byzantine Fault Tolerance (BFT) corresponds to the capacity of a system to tolerate failures against the Byzantine Generals Problem (BGP) [PSL80]. For example, consider an agreement scenario among a set of nodes where each one holds a possibly different initial value. All the nodes must agree on a single value by following a consensus protocol. Then, we consider a system to be Byzantine Fault Tolerant if most nodes reach an agreement even when a minority of the players are malicious and may diverge from the protocol arbitrarily.

The first solution for BFT was introduced by Lamport, Shostak, and Pease in 1982 [PSL80]. Since then, many Byzantine fault-tolerant algorithms and protocols have proposed to help resolve many of the misconceptions associated with Byzantine faults and the difficulties in preventing the propagation of related faults. In 1999 Miguel Castro and Barabra Liskov [CL99] showed that it is possible to implement a high-performance state machine replication in an asynchronous environment based on BFT. This protocol is called Practical Byzantine Fault Tolerance (PBFT) and achieves millisecond increases in latency processing thousands of requests per second. The algorithm organises nodes where the machine state replication will be held in replicas identified as $R = (0, \ldots, |R| - 1)$. The number of replicas $|R|$ is equal to $3f + 1$, where f is the maximum number of faulty replicas. Even though it is possible to have more replicas, additional ones can generate a downgrade in the system performance. The organisation of the replicas are called the views. In a view, just one of the replicas is considered as the primary, and the rest are the backups. The primary of a view is the replica p and is defined as $p = v \mod |R|$, where v corresponds to the view number. Views are changed when a failure in the primary is detected.

A general overview of the steps executed by the protocol is presented below:

1. A client sends a request to invoke the primary
2. The primary p broadcast the request to the backups
3. Replicas execute the request and send a reply to the client
4. The client waits for $f + 1$ replies from different replicas that have the same result
5. Once the $f + 1$ is reached, the clients accept the result as correct (final value)

All the replicas must to be deterministic and start in the same state in order to ensure the protocol's safety.

One implementation of PBFT in Blockchain is the consensus protocol called Sumeragi used by Iroha [Iro21]. The algorithm is based on BChain [DMPZ14] and it uses $2f + 1$ signatures to validate a transaction. The leader verifies the transaction, puts it in a queue and then signs the transaction. After this process is done, the leader sends the transaction to the remaining $2f+1$ peers for their validation. Finally, the consensus is reached once the transaction has the $2f + 1$ validations.

Another recent Byzantine consensus protocol is AlgoRAND [GHM+17]. This protocol has a novel property called player replaceability. This property guarantees security in an adversarial environment. Instead of using all the peers in the system, this protocol chooses a subset of peers by an algorithm called cryptographic sortition, which is a random process of selecting officials from a large set of eligible peers. Provided that honest users retain a fraction greater than 2/3 of the stakes. Three of the main characteristics of AlgoRAND:

- There are no forks that arise with an overwhelming probability.
- It only requires a minimal amount of computation.
- It reaches a consensus quickly.

1.3.4 Proof-of-Burn

In 2012, Iain Stewart proposed the first proof-of-burn model [Ste12]. It consists of a mechanism for the irrevocable and provable destruction of cryptocurrencies. In this type of algorithm, the miners show proof that they have burned some coins - that is, they sent them to a verifiably unspendable address (e.g.. Slimcoin [P4T14]). This is expensive from its individual point of view, as is Proof-of-Work, but it consumes no more resources than the underlying asset burned. Currently, all proof of burn works by burning proof-of-work-mined tokens, so the ultimate source of scarcity remains the proof-of-work-mined "fuel". It can also be used to bootstrap one cryptocurrency from another.

Chapter 2

Introduction to Modern Cryptography

Contents

Cryptography is a discipline that has been part of our society since antiquity. The earliest known use of cryptography is found in non-standard hieroglyphs carved into the wall of a tomb from the Old Kingdom of Egypt circa 1900 BC [Ltd06]. However, scientists think they meant to be for amusement rather than message protection or secure communication. Later in Mesopotamia, some clay tables dated near 1500 BC were found to encrypt a craftsman's recipe for pottery glaze, presumably commercially valuable [Kah96]. In ancient Greece, the spartan military used the scytale transposition cipher [Coh95]. Moreover, during the Roman Empire, they also used cryptographic techniques like the Caesar cipher [Kah96].

The first evidence of modern cryptography has been found among the Arabs, which documented the first cryptanalytic methods. The *Book of Cryptographic Messages*, wrote by Al-Khali, contains the first use of combinations and permutations for listing all the possible Arabic words with and without vowels [Bro11]. In the 9th century, Al-Kindi invented the frequency analysis technique to break mono-alphabetic substitutions ciphers mechanisms. This invention was the most relevant contribution to the cryptanalytic field until World War II.

In England, between the end of the 8th and the 11th century, the ciphering mechanism by substitution was frequently used by scribes to encipher notes, solutions to riddles, and colophons. This period saw vital and significant cryptographic experimentation in the West. In the 15th century, Leon Battista Alberti formalised the polyalphabetic cipher. This ciphering mechanism was tolerant to the cryptanalytic technique of frequency analysis used by these days. During the Kingdom of Louis XIV, the cipher method called "Great Cipher" developed by the family of French cryptographers, the Rossignols, was used.

By the end of the 20th century, the way to conceive cryptography has radically changed. It is the beginning of Modern cryptography that has arisen based on a new rich mathematical background that enables the study of cryptography as a science. This allows us to use cryptography for purposes broader than secret communication. For example, thanks to modern cryptography it is possible to do message authentication, digital signatures, protocols for exchanging secret keys, authentication protocols, electronic voting and digital cash.

This chapter covers the introduction to Modern Cryptography, starting with the principles and main properties in Section 2.1 and then the mathematical background that gives security to the modern cryptology in Section 2.2. Then, it is introduced the cryptographic primitives used in this thesis for Hashing (Section 2.3), Public Key Encryption and Digital Signatures (Section 2.4).

2.1 The Principles

Modern cryptography relies on three main principles: Definitions, Schemes and Proofs.

2.1.1 Definitions

The definitions are one of the key contributions of modern cryptography. The formalisation of the security properties is crucial in designing, using, and studying any cryptographic protocol. Once we are designing a protocol that is not enough to construct a secure cryptographic protocol intuitively, it is crucial to define the security properties that we want to achieve to evaluate the quality of our building. Having first formally what is needed will lead us to create the right thing.

Definitions are also important regarding the usage. For example, we use it embedded in a more extensive system wje, we build a cryptographic protocol. Hence, it is possible to verify whether the definitions satisfied by a given protocol are sufficient for the purpose of the system with clear definitions.

The definitions will also help to study the protocols from the security point of view. Based on the security definitions, we can analyse and compare if a protocol is preferable rather than another one according to the specific need of a larger system. Without security definition, the only analysis that we can do is the performance, which is not enough for security applications.

The formal definitions must also specify the attack model. For example, whether the attack used for an adversary will be a chosen-ciphertext attack (CCA) or a chosen-plaintext attack (CPA). Therefore, to fully define the security of some cryptographic protocol, two distinct issues must be explicitly addressed: the definition of break and the adversary's power. Regarding the break, an encryption scheme is considered broken if an adversary can learn some information of the plaintext from a ciphertext. Concerning the power of the adversary, this is related to the assumptions of the actions that the adversary is assumed able to take, as well as the adversary's computational power.

Finally, Jonathan Katz and Yehuda Lindell propose that any security definition must have the following structure [KL20]:

> A cryptographic scheme for a given task is secure if no adversary of a specified power can achieve a specified break.

2.1.2 Schemes

The security in most modern cryptographic constructions depends on the design of the schemes and the security assumptions. Hence, no protocol can be unconditionally proven secure. This impossibility is because their existence relies on problems of computational complexity that have not been answered yet. Considering that security relies upon some assumptions; therefore, the schemes must be designed according to a well-defined security assumption.

The security of the scheme depends on the assumption strength. Hence, for having a strong assumption, it must be studied. To strengthen it, the study must be done under a strong theoretical background; however, this is possible if and only if the assumption is clearly formalised. If the assumption relied on not precisely stated definitions, it could not be studied and eventually refuted. Hence, a precondition to increase our confidence in an assumption is having a precise statement of what exactly is assumed.

The design of a scheme based on explicit and well-defined assumptions will also help evaluate the security between different construction. Hence, assuming both schemes are equally efficient, If one scheme is based on weaker assumptions than the second one, then the first scheme is preferred since it may turn out that the second assumption is false while the first assumption is true. On the other hand, if the assumptions used by the two schemes are incomparable. In that case, the general rule is to prefer the scheme based on the better-studied assumption.

2.1.3 Proofs

In modern cryptography, a proof is done by using the reductionist approach. For example, this mechanism is used to prove a theorem of the form [KL20]:

> Given that assumption X is true, construction Y is secure according to the given definition

This shows how to reduce the problem given by an assumption X to the problem of breaking construction Y. The proof will typically establish (via a constructive argument) how any adversary that breaks the construction Y can be used as a subroutine to violate Assumption X.

2.2 Hard problems

As introduced in the previous section, clear and well-defined assumptions are necessary for modern cryptography to define the scheme's security. In order to prove security, we will use complex mathematical problems applied to cryptography as a basic building block for the scheme's security.

This subsection will introduce the main cryptographic hard problems used to prove security in the cryptographic primitives utilised in this thesis for hashing, digital signatures and asymmetric encryption schemes.

2.2.1 One-way Functions and Permutations

The one-way function is a basic block for constructing cryptographic hash functions and for secure signature schemes. A one-way function can be defined informally as a function f that is "easy" to compute and "hard" to invert. Thus, we can define a one-way function formally as follow:

Definition 2.1. *A tuple* $\Pi = (Gen, Samp, f)$ *of Probabilistic Polynomial Time (PPT) algorithms is a function family if* Π *holds the following:*

1. *Gen: is a probabilistic algorithm that takes as an input a security parameter* 1^k *and outputs parameters* I *with* $|I| \geq k$*. Each value of* I *defines sets* D_I *and* R_I *that corresponds to the domain and range of the function* f_I *defined below.*

2. *Samp: is a probabilistic algorithm that inputs the parameters I and outputs an element of D_I.*

3. *f: is a deterministic algorithm that takes as input parameters I and an element $x \in D_I$, and outputs an element $y \in R_I$ such that $y := f_I(x)$.*

Definition 2.2. Π *is a permutation family if holds Definition 2.1 and the following:*

1. *For all I output by Gen, the distribution defined by the output of $Samp(I)$ is a uniform distribution on D_I.*

2. *For all I output by Gen it holds $D_I = R_I$ and the function f_I is a bijection.*

If Π is a permutation family and there exists a polynomial p such that $D_I = \{0,1\}^{p(k)}$ for all I output by $Gen(1^k)$, then we say that Π is a permutation family over bit-strings.

Definition 2.3. *A function or permutation family $\Pi = (Gen, Samp, Eval)$ is one-way if for all PPT algorithms A there is a negligible function $\epsilon(\cdot)$ such that:*

$$\Pr\Big[I \leftarrow Gen(1^k); x \leftarrow Samp(I); y := f_I(x);\\ x' \leftarrow A(I, y) : f_I(x') = y\Big] = \epsilon(k)$$

Any one-way permutation family that satisfies with some additional conditions can be transformed into a one-way permutation family over bit-strings. Let Π be a one-way permutation family with $D_I \subseteq \{0,1\}^{p(k)}$ where p is a polynomial and I is the output of $Gen(1^k)$. Moreover, the following conditions must be held:

1. Given I, the set D_I is efficiently recognizable

2. For all I, the set D_I is dense in $\{0,1\}^{p(k)}$. That means $\frac{|D_I|}{2^{p(k)}} = \frac{1}{poly(k)}$

Now, let's construct a permutation family $\Pi' = (Gen', Samp', f')$, where Gen' is the same algorithm as Gen, $Samp'$ is an algorithm that outputs a random string of length $p(k)$, and the function $f'_I : \{0,1\}^{p(k)} \to \{0,1\}^{p(k)}$ is defined as:

$$f'_I(x) = \begin{cases} f_I(x) & x \in D_I \\ x & \text{otherwise} \end{cases}$$

We can note that Π' is not one-way because f'_I can be easily inverted at any point where $y \notin D_I$; however, it is difficult to invert in a fraction of its range. In other to overcome this limitation on the range, it is possible to amplify it by using many copies en parallel of Π'. Hence, we can define $\Pi' = (Gen'', Samp'', f'')'$ where Gen'' is the same algorithm as Gen, $Samp''$ is an algorithm that outputs a random string of length $l(k) \cdot p(k)$ for an appropriate polynomial l, and:

$$f''_I(x_1 || \dots || x_{l(k)}) \overset{def}{=} f'_I(x_1) || \dots || f'_I(x_{l(k)})$$

Now, using the construction Π'' it is easy to show that the inversion is hard as long as any of the $x_i \in D_I$.

2.2.2 Numbers Factorization

The number factorization problem is one of the most studied hard problems in number theory. This problem can be easily explained as a one-way function. Therefore, we can define the function $f_{mult}(x, y) = xy$ as a candidate for a one-way function. However, not just any factorization of numbers is necessarily hard, so it is needed to restrict the inputs of the function f_{mult} to consider the factorization to be a hard problem.

We can overcome this problem by restricting x and y to a large prime of equal length. We can define this using the same structure of Definition 2.1. Hence, let's define the family function $(Gen, Samp, f)$ as follows:

1. $Gen(1^k)$: is an algorithm that outputs $I = 1^k$ and D_I is the set of all pairs of k-bit primes.
2. $Samp(1^k)$: is a randomized algorithm that outputs two random and independently chosen k-bit primes.
3. $f(p, q)$ is an algorithm that outputs the product pq.

To state that the number factorization is a hard problem, we use the assumption that the family function $(Gen, Samp, f)$ is a one-way function.

Additionally, for the random prime generation process in polynomial time, we consider that the algorithm $Samp$ follows:

1. The random number generation process will be executed until obtaining, with high probability, a prime number.
2. There is a probabilistic polynomial-time algorithm that can determine whether a given integer is prime.

Let's define $GenModulus$ as a polynomial-time algorithm with input 1^k and outputs (N, p, q) such that $N = pq$, p and q are k-bit primes. Then, we can express the factoring assumption relative to the algorithm $GenModulus$ as:

Definition 2.4. *The number factorization is hard relative to GenModulus if for all PPT algorithm A there is a negligible function $\epsilon(\cdot)$ such as:*

$$\Pr\left[(N,p,q) \leftarrow GenModulus(1^k); (p,q) \leftarrow A(N) : pq = N\right] = \epsilon(k)$$

The number factorization assumption seems only to guarantee the existence of a one-way function. Hence, we will show that if factorize N is a hard problem, the factorization of N^2 is a one-way function.

2.2.3 Discrete Logarithm Problem

The discrete logarithm assumption is another hard problem used to build cryptographic schemes. In general, we will consider only groups $\mathbb{G}$ of primer order q. In such groups, all the elements belonging to these groups $\mathbb{G}$ (other than the identity) are generators. So, let's be g a generator of the group, and $h \in \mathbb{G}$ be arbitrary, the discrete logarithm of h concerning g, $log_g(h)$, is the smallest non-negative integer x such that $g^x = h$. Thus, the problem is to obtain x given g and a random group element h.

For certain classes of groups, the discrete logarithm problem is not hard to solve. Hence, the hardness of the problem depends on how the elements of the group are represented.

To formalise the discrete logarithm assumption, let $\mathcal{G}$ be a PPT algorithm that on inputs 1^k, outputs a cyclic group $\mathbb{G}$ with order q and a generator $g \in \mathbb{G}$.

Definition 2.5. *The discrete logarithm problem is hard with respect to $\mathcal{G}$ if for all PPT algorithm A there is a negligible function $\epsilon(k)$ such as:*

$$\Pr\left[(\mathbb{G},q,g) \leftarrow \mathcal{G}(1^k); h \leftarrow \mathbb{G}; x \leftarrow A(1^k,\mathbb{G},q,g,h) : g^x = h\right]$$

From the definition of the discrete logarithm assumption, it is possible to imply the existence of a one-way function family $f_{\mathbb{G},q,g}(x)$ that outputs g^x. The functions in this family are one-to-one if we take the domain of $f_{\mathbb{G},q,g}(x)$ to be $\mathbb{Z}_q$. For certain groups $\mathbb{G}$, we also obtain a one-way permutation family; for example, the groups of the form $\mathbb{Z}_q$ for p prime. In this case, the mapping $f_{\mathbb{G},q,g} : \mathbb{Z}_{p-1} \to \mathbb{Z}_p^*$ is one used above, where $f_{\mathbb{G},q,g}(x) = g^x \mod p$. Nevertheless, if we shift the domain, we get the function $f'_{\mathbb{G},q,g} : \mathbb{Z}_p^* \to \mathbb{Z}_p^*$:

$$f'_{\mathbb{G},q,g} = g^{x-1} \mod p$$

2.3 Hash Functions

Hash functions have a central role in the construction of secure cryptographic schemes applied to message integrity, digital signatures, password verification, blockchain, among others. The construction of a hash function often is based on well-known mathematical assumptions; nevertheless, we also have extremely efficient constructions (not based on any particular hard cryptographic problem) that one can reasonably assume to be secure. This subsection introduces the principal security definitions of a hash function and the Merkle-Damgard construction.

2.3.1 Definitions

Roughly speaking, a hash function can be described as a function that transforms an input into its compressed version. The hash functions of our interest are keyed, which means they have the form of $H_s(\cdot)$, where H is a hash family, and s is the key generated uniformly at random. From the security perspective, we are interested in two main properties: the collision-resistant and the universal one-way. The collision-resistant means that it is hard to distinct x and x' for which $H_s(x) = H_s(x')$. On the other hand, the universal one-way property means that it is hard to find a collision for a pre-specified input x, chosen independently of the key s.

Let's formally define the concepts introduced above:

Definition 2.6. *A hash function is a pair of PPT algorithm (Gen, H) such that:*

1. *Gen: is a probabilistic algorithm that on input 1^k outputs a key s*
2. *There exists a polynomial l such that H on input key s and $x \in \{0,1\}^*$, outputs a string $H_s(x) \in \{0,1\}^{l(k)}$*

Definition 2.7. *A Hash function (Gen,H) is a collision-resistant if there is a negligible function $\epsilon(k)$ such that:*

$$\Pr\left[s \leftarrow Gen(1^k); (x, x') \leftarrow A(1^k, s) : x \neq x' \land H_s(x) = H_s(x')\right]$$

Definition 2.8. *The pair (Gen,H) is universal one-way (second pre-image resistant) if there is a negligible function $\epsilon(k)$ such that:*

$$\Pr\left[x \leftarrow A(1^k); s \leftarrow Gen(1^k); x' \leftarrow A(1^k, s) : x \neq x' \land H_s(x) = H_s(x')\right]$$

Hence, we can easily see that the collision-resistance property implies universal one-way.

2.3.2 The Merkle-Damgåard Construction

The Merkle-Damgåard (MD) construction is one of the most used mechanisms for building hash functions. It consists in transform a fixed-length has a function (Gen, H') into a collision-resistance hash function (Gen, H) that accepts arbitrary length inputs.

To define the Merkle-Damgåard construction, we fist define the hash function (Gen, H') with inputs $2l(k)$. Then, to construct the hash function (Gen, H) the following must be held:

- The key generation algorithm Gen rest immutable
- H_s is defined for inputs of length at most $2^l - 1$.

Then, to compute $H_s(x)$ using the MD construction, we must do:

1. Set $L := |x|$ and $B := \lceil \frac{L}{l} \rceil$. Pad the input of x with zeros until its length is an integer multiple of l, and parse the result as sequence of l-bit blocks $x_1, \ldots, x_B$. Set $x_{B+1} := L$, where L is encoded using l bits
2. Set $z_0 := 0^l$
3. For $i = 1, \ldots, B + 1$, compute $z_i := H'_s(z_{i-1}$
   $vert|x_i)$
4. Output Z_{B+1}

2.4 Public Key Cryptography and Digital Signature

Public key cryptography enables parties to communicate privately without agreeing on any secret information in advance (like in private-key cryptography). Communication privacy is achieved thanks to public-key encryption, which is one instance of public-key cryptography. Nowadays, it is used daily in different applications like client-server communication, emails, among others. Moreover, it is possible to construct digital signature schemes for messages and documents authentication based on public-key cryptography.

This subsection will cover the basic principles and the main security properties of public-key encryption and digital signatures.

2.4.1 Public Key Encryption Schemes

Using public-key cryptography, we can define schemes for data privacy using a set-up based on a key pair (pk, sk), where p_k is the public key, and sk is the private key. Then, we can encrypt a message m using the public key p_k and thus obtain a ciphertext c to be shared with the message recipient, who keeps the private secret sk. After receiving the message c, the recipient decrypts c using the secret key sk to recover the original message.

The formal definition of public-key encryption included security parameter k that was not mentioned in the previous example. The security parameter k provides a way to study the asymptotic behaviour of a scheme. We need our algorithms to run in time polynomial in k, and our schemes offer protection against attacks that can be implemented in time polynomial in k. Moreover, we measure the probability of an adversary succeed in any attack in terms of k, and will require that any attack (that can be carried out in polynomial time) be successful with probability at most negligible also in k. Therefore we can consider the security parameter as an indication of the level of security offered by the scheme.

Definition 2.9. *A public-key encryption scheme consists of a tuple (Gen, Enc, Dec) that satisfies:*

1. *Gen: is the key generation algorithm that inputs the security parameter k and outputs a key pair (pk, sk), where pk is the public key and sk is the private key.*
2. *Enc: is the encryption algorithm that takes as input the public key pk and the message m and outputs a ciphertext c*
3. *Dec: is the deterministic decryption algorithm that takes as input the private key sk and the ciphertext c and outputs the clear message m or $\perp$ in case of failure.*

Moreover, the public-key encryption scheme (Gen, Enc, Dec) must satisfy that for all security parameter k and all key pair (pk, sk) the following equation:

$$m = Dec_{sk}(Enc_{pk}(m))$$

To consider the public key encryption scheme as secure, we need to define the security properties to be evaluated. This section covers the indistinguishability, security for multiple encryptions and security against chosen-chipertext attacks.

Indistinguishability: A public-key encryption scheme $\Pi = (Gen, Enc, Dec)$ is secure in terms of indistinguishability if a computationally bounded adversary (in polynomial time) can learn something about m with negligible probability. We define this in terms of a game $PubKey^{ind}_{A,\Pi}(k)$, where there is an adversary A such that:

1. $(Gen(1^k)$ is run to get the key pair (pk, sk) to then give the public key pk to A. The adversary A also gets access to an encryption oracle $Enc_{pk}(\cdot)$

2. A outputs two messages m_0 and m_1 of equal lengths.

3. A random bit b is chosen, and then the A request the ciphertext $c \leftarrow Enc_{pk}(m_b)$.

4. A outputs a bit b', and then if $b' = b$ we say that A succeeds.

Definition 2.10. *A public-key encryption scheme $\Pi = (Gen, Enc, Dec)$ is secure in the sense of indistinguishability if for all PPT adversary A there is a negligible function $\epsilon(k)$ such that:*

$$\Pr\left[PubKey^{ind}_{A,\Pi}(k) = 1\right] \leq \frac{1}{2} + \epsilon(k)$$

where the probability is taken over the random coins used by A and the random coins used to generate (pk, sk).

Security for multiple encryptions: A public-key encryption scheme $\Pi = (Gen, Enc, Dec)$ is secure in terms of multiple encryptions if an adversary (computationally bounded in polynomial time) can learn something about a message m after multiple encryption processes using the same key with negligible probability. We define the game $PubKey^{mult}_{A,\Pi}(k)$, where there is an adversary A such that:

1. $(Gen(1^k)$ is run to get the key pair (pk, sk) to then give the public key pk to A. The adversary A also gets access to an encryption oracle $Enc_{pk}(\cdot)$

2. A outputs two arrays of messages $M_0 = (m_{0,1}, \ldots, m_{0,t})$ and $M_1 = (m_{1,1}, \ldots, m_{1,t})$ with the messages $m_{0,i}$ and $m_{1,i}$ of equal lengths for all i

3. A random bit b is chosen, and then A request the array of ciphertext $C = (Enc_{pk}(m_{b,1}, \ldots, Enc_{pk}(m_{b,t})$

4. A outputs a bit b', and then if $b' = b$ we say that A succeeds.

Definition 2.11. *A public-key encryption scheme $\Pi = (Gen, Enc, Dec)$ is secure in the sense of multiple encryptions if for all PPT adversary A there is a negligible function $\epsilon(k)$ such that:*

$$\Pr\left[PubKey^{mult}_{A,\Pi}(k) = 1\right] \leq \frac{1}{2} + \epsilon(k)$$

Security for chosen-ciphertext attacks: The security under chose-ciphertext attacks is relevant because, in this scenario, the adversary A has an active position querying the decryption oracle with some chosen-ciphertext c', which may depend on the original ciphertext c, to learn something about the original message m. Thus, a public-key encryption scheme $\Pi = (Gen, Enc, Dec)$ is secure against chosen-ciphertext attacks if an adversary (computationally bounded in polynomial time) can learn something about a message m after repetitively requesting decryption of ciphertexts chosen by him with negligible probability. We define the game $PubKey^{cca}_{A,\Pi}(k)$, where there is an adversary A such that:

1. $(Gen(1^k)$ is run to get the key pair (pk, sk) to then give the public key pk to A. The adversary A also gets access to the encryption oracle $Enc_{pk}(\cdot)$ and the decryption oracle $Dec_{sk}(cdot)$.
2. A outputs two messages m_0 and m_1 of equal lengths.
3. A random bit b is chosen, and then the A request the ciphertext $c \leftarrow Enc_{pk}(m_b)$.
4. A can repeatedly request the decryption of any ciphertext c' of its choice (except for c), obtaining $m' \leftarrow Dec_{sk}(c')$.
5. A outputs a bit b', and then if $b' = b$ we say that A succeeds.

Definition 2.12. *A public-key encryption scheme $\Pi = (Gen, Enc, Dec)$ is secure against chosen-ciphertext attacks if for all PPT adversary A there is a negligible function $\epsilon(k)$ such that:*

$$\Pr\left[PubKey^{ccp}_{A,\Pi}(k) = 1\right] \leq \frac{1}{2} + \epsilon(k)$$

2.4.2 Digital Signature Schemes

Digital signatures are another application of public-key cryptography. Its application is broad, and we can find it in message authentication, data integrity, among others. For example, we have a software vendor who has a key pair (pk, sk). On the contrary to public-key encryption, the communication flow goes in the other way (from the vendor to a user). Specifically, when the vendor wants to send a message m (i.e., a software update) in an authenticated manner to the user. The vendor will use its secret key to sign the message and compute a signature σ. Then the message and its signature are transmitted to the user. Upon the user receives (m, σ), he can utilize the vendors public key to verify that σ is a valid signature on m.

Let's define a signature scheme as a tuple of $(Gen, Sign, Ver)$, where Gen is the key generation algorithm with security parameter k, $Sign$ the signature algorithm, and Ver the algorithm to verify the signature. Formally, we can define this as:

Definition 2.13. *A signature scheme is a tuple of PPT algorithm (Gen, Sign, Ver) that satisfies:*

- *Gen: is the key generation algorithm inputs the security parameter k and outputs a key pair (pk, sk), where pk is the public key and sk is the private key.*
- *Sign: is the signing algorithm that takes as input a message m (from an underlying message space that may depend on pk) and a private key sk*
- *Ver: is a deterministic verification algorithm that takes as input a message m, a signature σ and a public key pk. It output a bit b with $b = 1$ if the signature is valid and $b = 0$ if it is invalid*

Moreover, the signature scheme $(Gen, Sign, Ver)$ must satisfy that for all security parameter k, all key pair (pk, sk) and all message m in the appropriate underlying plaintext space, the following equation:

$$1 = Ver_{pk}(Sign_{sk}(m))$$

Security of the signature scheme: One of the most crucial security properties for any digital signature scheme is the capacity to make extremely difficult to forge a signature. Formally, given a public key pk generated by a signer S, we say that a digital signature $\Pi = (Gen, Sign, Ver)$ is unforgeable under an adaptive chosen-message attack if an adversary (computationally bounded in polynomial time) can forge the digital signature of a message m (that has not being signed by S) after repetitively requesting the signature of many other messages chosen by him with negligible probability. We define the game $Sign - forge^{cma}_{A,\Pi}(k)$, where there is an adversary A such that:

- $(Gen(1^k)$ is run to get the key pair (pk, sk) to then give the public key pk to A. The adversary A also gets access to a signing oracle $Sign_{sk}(\cdot)$
- A outputs (m, σ).
- Lt $\mathcal{Q}$ be the set of messages whose signatures were requested by A during its execution. We say that A succeed if $m \notin \mathcal{Q}$ and $Ver_{pk}(m, \sigma) = 1$

Definition 2.14. *A public-key encryption scheme $\Pi = (Gen, Sign, Ver)$ is existentially unforgeable under adaptive chosen-message attack if for all PPT adversary A there is a negligible function $\epsilon(k)$ such that:*

$$\Pr\left[Sign - forge^{cma}_{A,\Pi}(k)\right] \leq \frac{1}{2} + \epsilon(k)$$

Part II

Privacy Preserving Distributed Protocols

Chapter 3

BlindCons: A Consensus Algorithm for Privacy-Preserving Private Blockchains

Contents

Blockchains are becoming gradually more sought-after in many organisations. Predominately because they allow organisations to overpass the traditional limitations of public blockchains regarding energy consumption, efficiency, and system control. However, such systems are in some ways losing their blockchain essence, particularly their unique feature of not being controlled by a central authority that governs many functionalities of the chain, including user enrolment. Our main contribution in this work is to propose a privacy-preserving user credential consensus algorithm for private blockchains, called *BlindCons*. The main security properties of our consensus protocol BlindCons uses an abstract model for blind signatures schemes and a Byzantine Fault Tolerance (BFT) consensus for the blockchain replication process. Our protocol guarantees the consistency of the data stored on the blockchain, the system's liveness, and users' privacy by blinding their signatures to keep the user's identity private. In addition, BlindCons guarantees user-transaction unlinkability, which means that it is impossible to link a transaction to a user using the data stored in the blockchain.

3.1 Introduction

Blockchain technology has gained enough popularity and was widely adopted in many fields such as finance, health, logistics, eVoting, among others; this is thanks to its immutability and transparency properties. However, Public Blockchains cannot manage tailored business rules like special data access restrictions and user profiling.

Many groups of companies have proposed *Private Blockchains* or *Permissioned Blockchains*, like for instance, the MyHealthMyData consortium [Con16a], which aims to connect hospitals and research centers across Europe to share medical data through a private Blockchain network. These Blockchains aim to manage data storage on a few nodes controlled by a company or a consortium in order to validate the transactions faster, consume less energy and reduce minning cost. Another example is PlasticTwis projectt [Con16b] which creates a new circular economy based on an ERC20 token implemented on a Permissioned ledger to promote plastic recycling in Europe. Moreover, Maersk and IBM have also developed TradeLens [IBM16], a supply chain system supported by the Permissioned Blockchain Hyperledger Fabric. S. Furthermore, blockchain use cases are increasing due to the adoption of Blockchain in closed ecosystems and enterprise applications. As a result, the European Blockchain Observatory and Forum has published a technical report [TL18] recommending private or permissioned Blockchains for storing sensitive data.

Although Permissioned Blockchains have several features suitable for sensitive data services, they have drawbacks related to transactions and user linkability. We propose using blind signature schemes to sign the transactions without linking user information to overcome this drawback. This model allows validating transactions without exposing the users identification, and thus maintaining his privacy.

Contributions: In this work, we aim at designing a new privacy-preserving consensus algorithm that ensures the unlinkability between a transaction recorded in a Permissioned Blockchain and the user that generated it. Our contributions are:

- A privacy-preserving consensus mechanism called *BlindCons* for Permissioned Blockchains.
- The abstract models for the blind signature scheme and the consensus algorithm.
- A consensus protocol that ensures the transaction unlinkability and the Byzantine Fault Tolerance (BFT) algorithm for the generation and commitment of new blocks.

Moreover, our construction is generic as it can be instanciated by any blind signature that satisfies the classical properties of such primitives.

Related Work: Blockchain platforms such as Bitcoin [Nak08] and Ethereum [Woo14] are popular because of the "anonymity" offered by their cryptocurrencies. However, Bitcoin still exhibits problems of transaction linkability and traceability. Over the past few years, some improvements have been proposed to increase Bitcoin's anonymity [SMD14], and cryptocurrencies have also been developed to overcome these issues, such as Zcash and Monero. For example, in Zcash [HBHW16], the protocol achieves anonymity by using ZK-SNARKs as cryptographic proof. In the same line, Bulletproofs [BBB$^+$18] proposes a NIZKP protocol with short proofs without relying on a trusted setup for confidential transactions or privacy-preserving smart contracts. On the other hand, Monero is a protocol based on the Cryptonote [VS13] that achieves unlinkability and untraceability by using a one-time random address and ring signatures. Although all these protocols aim to improve the privacy of the transactions, they are designed for Permissionless Blockchains, where the transaction linkability differs from permissioned architecture where the users are not anonymous. Hence, there is a strong link between the transaction signature and the user who is triggering it.

A different approach to ensure anonymity is the eCash [Cha82, CFN88] model proposed for Bitcoin in [HBG16]. However, the idea to use a third party to generate a blind voucher is not aligned with the principle of decentralisation that Bitcoin or any other Permissionless Blockchains have.

lthough several papers have been published on Blockchain privacy for permissionless ledgers, to the best of our knowledge, there is no formal protocol that addresses the transaction linkability issue in Permissioned ledgers.

Outline: In Section 3.2, we present an introduction of permissioned blockchains, an abstraction for blind signature schemes and their security notions. In Section 3.3, we introduce the BFT consensus mechanism. Then, in Section 3.4, we describe our privacy-preserving consensus algorithm BlindCons. In Section 3.5, we explicit the blockchain security properties of BlindCons, and finally we conclude in Section 3.6.

3.2 Preliminaries

We start by introducing permissioned blockchains, and then we present an abstract model for blind signatures schemes and their security notions.

3.2.1 Permissioned Blockchain

Blockchain technology has evolved from the permissionless architecture for cryptocurrencies implementation to permissioned models for business applications. Blockchain technology is particularly interesting for business purposes such as supply chain, process traceability, secure logs, among others. However, an open platform with anonymous users is not entirely suitable for general business requirements. Hence, permissioned blockchains receive attention from different industries, due to their distributed ledger capability and user access restrictions.

Permissioned blockchain such as Tendermint [Inc20], Multichain [Sci20] and Hyperledger [Fou19] rely on different consensus mechanisms than permissionless blockchain. TThese blockchain technologies' machine state replication process is based on Byzantine Fault Tolerant (BFT) protocols rather than expensive mechanisms to prove honesty like Proof-of-Work used in Bitcoin and Ethereum. The protocols based on BFT use the assumption that having $3f+1$ nodes, where f is the number of faulty nodes, the network can reach a consensus. The mechanism considers that each node validates and votes for the new block proposals, and when there are enough valid answers for the new block, the network considers the block as valid and appends it to their local version of the blockchain. Considering that the network's reliability depends on the number of faulty nodes f, the users or the nodes access to the network becomes crucial.

The user access is managed in different ways according to the permissioned blockchain to be used. For example, Multichain uses administrators to manage the user's access [Gre15]. These administrators link and grant the user credentials with specific privileges for transacting or mining within the network. At the same time, the Hyperldger Fabric [ABB$^+$18], uses certificate authorities across the network to enrol the users and a specific smart contract or chaincode policies to restrict some users operations.

The advantages of permissioned architecture for a closed environment or federated networks contribute to increasing the popularity of blockchain for new applications. However, the strict user regulations (i.e., enrolment and user privileges) produce a strong transaction-user linkability issue, which is seen as a drawback from the identity privacy perspective.

3.2.2 Blind Signature Model

A Blind Signature scheme allows a user U to obtain a blind signature of the message $M \in \{0,1\}^*$ issued by an authority (A).

Definition 3.1 (Blind Signature scheme [Cha83]). *A blind signature* scheme *is a Probabilistic Polynomial Time (PPT) algorithm, which consists of three protocols organised as follows:*

Key Generation: *$\mathsf{KeyGen}(1^k)$ is a probabilistic algorithm that uses as input the security parameter k and it returns a key pair (pk,sk).*

Blind signature: *$\mathsf{BlindSign}(\mathcal{U}(M,\mathsf{pk}),\mathcal{A}(\mathsf{sk}))$ is a protocol run by the user U and the authority A. $\mathcal{U}$ uses as input the message M and the public key pk, and $\mathcal{A}$ gets the private key sk as input. After running the protocol, we obtain an output σ, which corresponds to the signature of M, or $\perp$ if the protocol ended unsuccessfully.*

Verification: *$\mathsf{VerifyBlindSign}(M,\sigma,\mathsf{pk})$ is a deterministic protocol that given a message $M \in \{0,1\}^*$, a signature σ, and a public key pk returns 1 if σ is a valid signature of M with regards to pk, otherwise it returns 0.*

The blind signature model presented in this section is suitable for private blockchain architecture due to the blinding process is performed by the same authority responsible for the enrolment process, where the authority A blinds the signature of the message proposed by the user U.

3.2.3 Security Notions for Blind Signature Schemes

A secure blind signature scheme must satisfy with three main properties: *Correctness*, *Unforgeability* and *Blindness.*

Definition 3.2 (Correctness). *A Blind Signature scheme is* correct *if for all $M \in \{0,1\}^*$, the user U and the authority A follows the protocol (pk,sk)$\leftarrow$ $\mathsf{KeyGen}(1^k)$ for all $k \in \mathbb{N}$ and $\sigma \leftarrow \mathsf{BlindSign}(\mathcal{U}(M,\mathsf{pk}),\mathcal{A}(\mathsf{sk}))$, and it holds $\mathsf{VerifyBlindSign}(M,\sigma,\mathsf{pk})$=1.*

Definition 3.3 (Unforgeability). *A blind signature scheme is* unforgeable *if for an honest authority A and adversary U^*, which has access to a PPT algorithm $\mathcal{U}^*$, there is a negligible[1] function $\epsilon(\cdot)$ such that:*

$$\Pr\left[\begin{array}{c} (pk,sk) \leftarrow \mathsf{KeyGen}(1^k) \\ (M_i^*,\sigma_i^*) \leftarrow U^{*(\mathsf{BlindSign}(\cdot,\mathcal{A}(\mathsf{sk})))}(\mathsf{pk}), i \in [1,\dots,t]) \end{array} : \begin{array}{c} M_i^* \neq M_j^* \; \forall i \neq j \; (i,j \in [1,\dots,t]) \; \wedge t > l \, \wedge \\ \mathsf{VerifyBlindSign}(M_i^*,\sigma_i^*,\mathsf{pk}) = 1 \; \forall i \in [1,\dots,t] \end{array}\right] = \epsilon(k)$$

[1]A function f is negligible, if for every positive polynomial P, $\exists K, \forall n > K, f(n) < \frac{1}{P(n)}$.

Where l is the number of executions of signature requests made by the adversary U^* using the PPT algorithm $\mathcal{U}^*$.

We use the blindness definition presented in the malicious signer model [Oka06, FHS15, ANN06]. The adversary is a malicious authority A^* with access to a PPT algorithm $\mathcal{A}^*$ and two honest user instances. Nevertheless, before defining blindness, we will introduce the following game:

1. The adversary A^* generates the public key pk and two messages M_0 and M_1.
2. We run a random selection of $b \in \{0, 1\}$.
3. The adversary A^* runs two instances of the signing protocol using M_b, $M_{\bar{b}}$ and the PPT algorithm $\mathcal{U}^*$, where $\bar{b} = 1 - b$ and $\mathcal{U}^*$ corresponds to the user's algorithm.
4. If we get $\sigma_0 = \perp$ or $\sigma_1 = \perp$ from the signing protocol, the adversary then A^* gets $(\perp, \perp) \leftarrow (\sigma_0, \sigma_1)$.
5. The adversary A^* outputs $b' \in \{0, 1\}$.

Definition 3.4 (Blindness). *A Blind signature scheme is blind if for all adversary A^* with access to a PPT algorithm $\mathcal{A}^*$ with access to two user instances, there is a negligible function $\epsilon(\cdot)$ such that:*

$$\Pr\left[\begin{array}{ll} b \overset{\$}{\leftarrow} \{0,1\}, & \\ (M_0, M_1, \alpha, \mathsf{pk}) \leftarrow \mathcal{A}^*(1^k), & \\ \alpha \leftarrow \mathcal{A}^{*\left(\mathsf{BlindSign}(\mathcal{U}(M_b, \mathsf{pk}), \cdot), (\mathsf{BlindSign}(\mathcal{U}(M_{\bar{b}}, \mathsf{pk}), \cdot)\right)}(\alpha), & : b^* = b \\ \textit{If } \sigma_0 == \perp \vee \sigma_1 == \perp \ \ \textit{then } (\sigma_0, \sigma_1) \leftarrow (\perp, \perp), & \\ b^* \leftarrow \mathcal{A}^*(\alpha, \sigma_0, \sigma_1) & \end{array}\right] - \frac{1}{2} \leq \epsilon(k)$$

Different constructions for blind signatures schemes satisfy these properties like the ones proposed by Chaum [Cha82], Schnorr [Sch91], Okamoto [Oka92, Oka06], Fuchsbauer et al. [FHS15], among others. Therefore, the user-transaction unlinkability model proposed in our protocol is generic, and it can be implemented by using different signature schemes that satisfy the security notions explained above.

3.3 Byzantine Fault Tolerant based Consensus for Permissioned Blockchain Architecture

The blockchain is a decentralised database organised in blocks that are appended one behind the other by using a hash pointer. Each block contains records that include the data to be stored in the chain. The blockchain design makes it suitable for a distributed ledger, due to the record organisation inside a block (i.e., financial transactions) and the hash chain between blocks that makes it resistant to malicious modifications. The blockchain uses a consensus algorithm to replicate the chain in each node member of the network. Nevertheless, the selection or design of the consensus algorithm must be consistent with the blockchain architecture and the openness of the system.

In permissioned blockchain architectures, every user must be enrolled into the system through Certificate Authorities (CA) or user administrators before joining the network. These authorities are responsible for generating the user credential for each new client or peer. This model provides a base of knowledge of the network members, making Practical Bizantine Fault Tolerance (PBFT) [CL99] or BFT-Smart [BSA14] based algorithms suitable for the blockchain replication process.

Our consensus approach for permissioned ledger has a different aim of the permissionless replication protocol. The security of the consensus algorithm in a permissionless architecture is achieved by proving the node's honesty by expending its resources (i.e., computing capacity, power consumption, stakes, among others). In contrast, a BFT-based protocol reaches the consensus accepting as valid the result proposed by the majority of the nodes.

Our consensus algorithm is based on BFT and the *execute-order-validate* process used in Hyperledger Fabric [ABB+18] since version 1.0. In this protocol, each transaction passes through these three stages.

1. The *execution* phase triggers the transaction and then it is validated by the validating peers.

2. In the *order* phase, the transactions are organised into a block according to the BFT consensus protocol.

3. Finally, the *validation* phase checks the security policies per operation and application type.

In the case of our BFT-based consensus algorithms, we start with a blockchain client c_{bc}, which is also a node member of the network with a local version of the blockchain. The client c_{bc} sends a transaction proposal with the instructions to be executed in the blockchain by the verifier peers (verifiers) to verify and validate the transaction. This verification process consists of the client signature authentication and the execution of the instruction that has been sent in the message. During the execution process, the verifier peers verify that the transaction has a correct instruction to be committed by the network. This process is analogous to the one executed by the primary in the original PBFT algorithm [CL99], which sends the message with the instructions to the backups to validate it. As for the transaction validation, the verifier peers vp_{bc} send their responses to the client, which collects them after the validation process is concluded. If the transaction is valid, the client c_{bc} sends the transaction to the consensus peers $consp_{bc}$ (consensors) to add it into the new block that is committed by the nodes' members of the network. The consensors are responsible for proposing and agreeing on the new block that shall be appended to the chain. This process is also homologous to PBFT, where one consensor is the leader, and the others are the backups. First, the consensor leader collects a group of transactions and organises them by order inside of a new block. Then the consensor leader sends a **propose** message with the block proposal to the consensors' backups to validate it. Each consensor backup validates the block proposal and responds to the rest of the consensors with a **prepare** message. Once the consensors have received enough valid **prepare** responses, they submit a **commit** message to the consensor, client and the nodes responsible for keeping a local version of the blockchain.

To exemplify this process, consider that we have a client c_{bc} that wants to execute an operation o_{bc} on our BFT-based blockchain protocol with verifier peers vp_{bc_i}, where $i = 1, ..., n$, and n is the number of the verifiers; and consensors peers $consp_{bc_i}$, where $i = 1, ..., k$, and k is the number of the consensors. To process this request, we need to execute the following four steps as presented in Figure 3.1:

1. Transaction Proposal,
2. Transaction Validation,
3. Broadcasting to Consensus,
4. Commitment.

Nevertheless, before we start describing the protocol, we introduce some notations used in this work. The information passes through the peers or nodes within the network in any distributed system by using messages. Hence, we denote a message by using $< \cdot, \dots, \cdot >$, where $\cdot$ corresponds to the message arguments. Moreover, we use the notation $(\cdot, \dots, \cdot)$ to group arguments inside a message.

1) ***Transaction Proposal:*** The client c_{bc} generates a message to execute specific operations o_{bc} to be resolved by the network. This message corresponds to a transaction proposal message to be sent to the verifier peers. The transaction proposal consists of invoking the operations o_{bc} and computing the state update and version dependency denoted by *stateUpdate* and *verDep*, respectively. The version dependencies relate the variables involve in the transaction with the local version of the variable in the client's node and their respective operations. For example: if a client wants to read and write in the blockchain, the version dependency is a tuple $(readset, writeset)$ where:

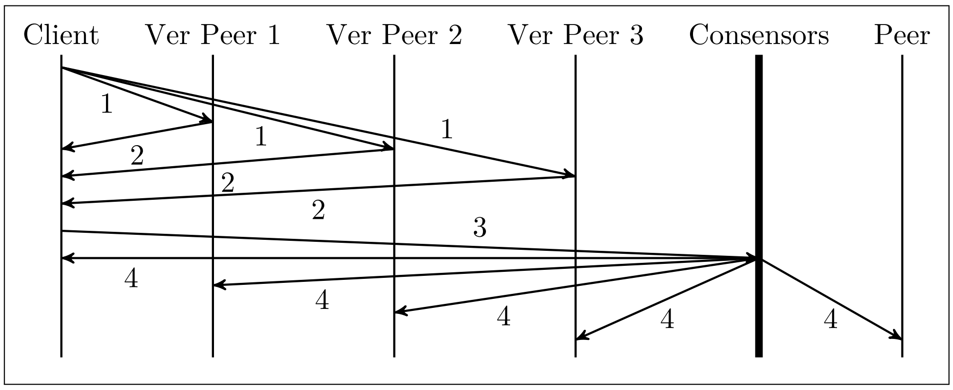


Figure 3.1 – BlindCons Transaction Flow.

- for every variable k read by the transaction, the pair $(k,\ s(k).version)$ is added to the *readset*,
- for every variable k modified by the transaction, the pair $(k, s(k).version)$ is added to the *writeset*.

Once the execution of the operations o_{bc} triggered by the client c_{bc} is completed, the client c_{bc} generates the *transaction proposal* message and then submits it to the verifier peers. The transaction proposal message is defined as:

$< proposal, trans_{prop} >$, where:

- $trans_{prop} := (c_{bc}, o_{bc}, txPayload, stateUpdate, verDep, retryFlag, \sigma_c)$.
- c_{bc} is the client ID,
- o_{bc} refers to the operations implemented in the distributed system,
- $txPayload$ is the payload of the submitted transaction,
- $retryFlag$ boolean variable to identify whether to retry the transaction submission in case of the transaction fails,
- σ_c is the client signature.

2) ***Transaction Validation:*** The verifier peers (vp_{bc_i}) verifies the client signature σ_c coming in the transaction proposal message. If the client signature is valid, the verifier simulates the transaction proposal by executing the operation o_{bc} with the corresponding *txPayload*, and then validates that the *stateUpdate* and *verDep* are correct. If the validation process is successful, the verifier peer generates a *transaction valid* message to be sent to the client c_{bc}. The message has the following structure:

$< \textit{transaction-valid}, tx_{id}, \sigma_{vp_i} >$, where:

- tx_{id} is the transaction identifier generated with the client ID and a nonce,
- σ_{vp_i} is the signature of the message signed by the verifier peer σ_{vp_i}.

In the case of the simulation process ending unsuccessfully, the verifier peer generates an *transaction invalid* message:

$< \textit{transaction-invalid}, tx_{id}, Error, \sigma_{vp_i} >$, where $Error$ can be:

- *incorrect-state*, when the validator obtains a different *state update* than the one coming in the *transaction proposal*,
- *incorrect-version*, when there is the newest version of the variable referred in the *transaction proposal*,
- *rejected*, for any other reason.

3) ***Broadcasting to Consensus:*** The client c_{bc} waits then for the response from the verifier peers. When it receives enough *Transaction Valid* messages adequately signed, the client stores the valid signatures into a package called *validations*. Once the transaction is considered validated, the client invokes the consensus services by using *broadcast(blob)*, where *blob:=(trans$_{prop}$, validations)*. Finally, the client c_{bc} sends a message to the consensor leader through the*broadcast* service to trigger the consensus process. The message has the following structure:

$< consensus, trans_{prop}, Validations >$, where:

- $Validations$ is the array with the signatures of the verifier peers that have validated the transaction.

The number of valid responses required to consider the *transaction proposal* as validated depends on the configuration of the permissioned blockchain. If the transaction has failed to collect enough validations, the client abandons this transaction proposal. On the other hand, if there is a problem with the verifier leader, the *broadcast* service submits the message to the next verifier in the list become the new leader.

4) ***Commitment:*** Once the client c_{bc} broadcasts the transaction properly validated to consensus, the consensus services collect this transaction and organise it into a block. The consensus process for the new block is performed by the consensors based on a Byzantine Fault Tolerant process (see Figure 3.2). The active consensors vote for the block validity and agrees on the new block based on the voting majority. The consensus service is sorted in *views*, where each *view* represents how the consensors are arranged for the consensus service. In each *view*, the consensor $consp_{leader}$ acts as the primary or leader, and the rest of the consensors $(consp_{bc_i}; i = 1, \ldots, k-1)$ correspond to the backups. In the case that the primary is faulty or is acting maliciously, the view changes and the next backup on the list turns in the new leader. The consensus process begins when the consensor leader, $consp_{leader}$, collects enough transactions to create a new block. The new block has the form: $B = ([(tx_1, validations_{tx_1}), (tx_2, validations_{tx_2}), \ldots, (tx_k, validations_{tx_k})], h)$, where h corresponds to the hash value of the previous block. The leader proposes this new block to the backups by sending a $propose_{block}$ message as follows:

$< propose_{block}, B, \sigma_{consp_{leader}} >$, where:

- B is the new block to be validate by the consensors.
- $\sigma_{consp_{leader}}$ is the signature of the $propose_{block}$ message sent by the consensor leader.

Then, each backup validates the $propose_{block}$ and responds to the other consensors of the service with the following message called $prepare_{block}$:

$< prepare_{block}, B, \sigma_{consp_i} >$, where:

- σ_{consp_i} is the signature of the $prepare_{block}$ message sent by the consensor $consp_{bc_i}$.

Once each backup has received enough valid $prepare_{block}$ responses, it will send a $commit_{block}$ message, structured a $< commit_{block}, B, \sigma_{consp_i} >$, to the nodes connected to the network to confirm that the block has been validated successfully. Otherwise, the backups response with an invalid message $< block - invalid\ , B, error, \sigma_{consp_i} >$.

From the consensus services, we get a hash-chained sequence of blocks with the valid transactions. The nodes receive the blocks from the consensus services through a direct connection or by using a gossip protocol. The nodes will wait until receiving $(50\% + 1)$ valid responses from the consensus service to consider that the block is ready to be committed. Once the peers have the confirmation that the new block is correct, they check if the validations of each transaction are valid according to the policy configured in the network. The peers then verify the *verDep* in order to ensure there are no conflicts between the operation o_{bc}; the variables involved in the transaction, and the current blockchain *state*. If this process finishes successfully, the transactions are then committed. On the other hand, if one of the validation fails, the peers consider the transaction as invalid, and it is dropped. Finally, the invalid transactions are informed to the client c_{bc}, and according to the

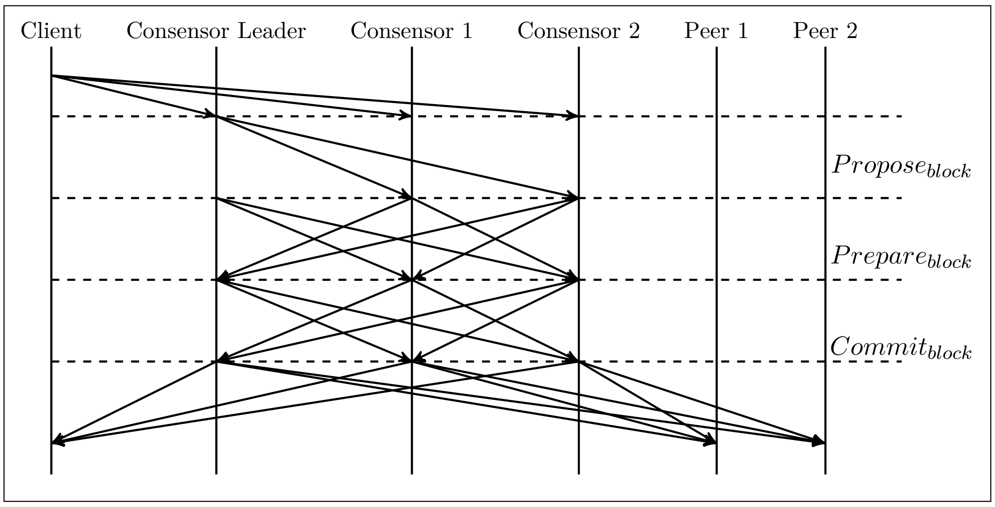


Figure 3.2 – BlindCons consensus service

retryFlag, the client may retry the transaction.

The nodes can change the local state only when the transactions are committed. Hence, each node connected to the network updates its local version of the ledger once all the transactions in the block are committed. Finally, the state update is done by appending the new block to their local blockchain version.

3.4 Privacy Preserving BFT for Permissioned Ledgers

Considering the permissioned BFT-based consensus protocol introduced in Section 3.3, we use the digital signature as a user authentication method, where each key pair used to sign a message is unique for each user. Moreover, each user is enrolled through one of the certificate authorities or user administrators belonging to the network. This cause a strong linkability issue between the users and the transactions, affecting the privacy level of the blockchain network. In order to overcome this issue, we propose to blind each user's signature by using the certificate authorities or the user administrators available in the network. Thus, the protocol follows the following steps:

1. Alice sends a newly signed transaction to one of the membership authorities (A),

2. Alices signature is validated only by A,

3. A signs the transaction proposed by Alice and anonymises Alices identity,

4. All the nodes member of the transactions' validation process can validate the A's signature,

5. A signature cannot be forged.

Now, to keep the client's and peers' privacy involved in the transactional process, we need to hide his ID and blind his signature. However, we do not address the ID hiding process with any particular mechanism. Therefore, we consider that the ID is replaced by a value corresponding to the anonymised user ID, and this process can be performed by using different schemes.

To address the issue related to the digital signature, we replace the signing mechanism used in the original protocol with the abstract model for the blind signature scheme introduced in Section 3.2.2. By recapitulating the consensus protocol above mentioned, the transactional process consists of the following steps:

1. **Transaction Proposal:** The client c_{bc} generates a signed message to execute an operation o_{bc} in the network.

2. **Transaction Validation:** The verifier peers vp_{bc} validate the client c_{bc} signature and verify that the transaction is correct by performing a simulation of the operation o_{bc} using his local version of the blockchain. Then, each verifier peer generates a signed transaction with the result of the validation process and sends it back to the client c_{bc}.

3. **Broadcasting to Consensus:** The client c_{bc} collects the *validations* coming from the verifier peers connected to the network. Once c_{bc} collects enough valid answers from the verifiers, it broadcasts the transaction proposal with the *validations* to the consensus service.

4. **Commitment:** All the transactions are ordered within a block and are validated with their own *validations*. The new block is then spread through the network to be committed by the peers.

To maintain consistency and liveness, we keep the transactional flow. However, the steps are modified to accept the new blind signature scheme to authenticate the clients and the peers.

We use the signing function presented in Section 3.2.2 and a new function representing the operation execution inside the blockchain protocol. Let $a||b$ be the concatenation operation between a and b, $\mathsf{BlindSign}((\mathcal{U}(M, \mathsf{pk}),(\mathcal{A}(\mathsf{sk}))$ and $\mathsf{VerifyBlindSign}(M,\sigma,\mathsf{pk})$ be the functions to generate a blind signature and to verify the blinded signature, respectively. Where M corresponds to the message to be signed, σ to the blinded signature; and pk to the public key. The result obtained from the function $\mathsf{BlindSign}$ corresponds to the blinded signature σ. On the other hand, the function $\mathsf{VerifyBlindSign}$ outputs a *valid* or *invalid* response with regards to M and pk. The third function corresponds to *EXEC(o,Payload)*, and represents the execution process of the operation o on the *Payload* performed by the peer. Additionally, we use the variables *tid* as the transaction ID, *SecurityPolicies* to define the set of security parameters configured in the node for the operation o_{bc} (i.e., read and write rights), and *TotVerPeers* for the total number of verifier peers in the network.

Now that the functions and the variables used in the protocol have been defined, we need to integrate them inside of each step of the transactional flow. The transactional process starts with the function **Transaction Proposal** (see Algorithm 1). In this step, the client c_{bc} executes the operation o_{bc} on the *Payload* and gets the *verDep* and *stateUpdate*. A random number then replaces the user ID, and we concatenate the arguments $(crand, o_{bc}, Payload, verDep,\ stateUpdate, retryFlag)$ to be signed. The signing process is presented in Figure 3.3). It is performed between the client and the authority A (e.g., the user administrators or the CAs) using the functions KeyGen and BlindSign that generate the key pairs and the blinded signature according to the protocol described in Section 3.2.2. This process is performed off-chain, hence the blind signature is generated before sending the transaction to the network. Moreover, the blind signature protocol can be executed by the client and multiple CAs or user administrators. Therefore, the signing process does not rely on a single authority to perform the blinding protocol. Finally, the client c_{bc} generates the *propose* message to be sent to the endorsing peers (see Figure 3.4).

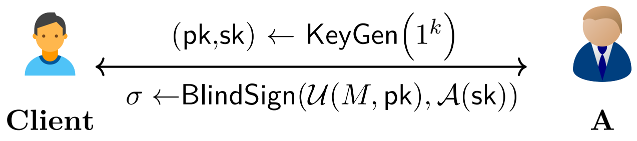


Figure 3.3 – Interactions between the client and A.

Algorithm 1 TxProp(o_{bc},*Payload*,*retryFlag*)

```
1: $(verDep, stateUpdate) \leftarrow EXEC(o_{bc}, Payload)$
2: $crand_{bc} \overset{r}{\leftarrow} \mathbb{N}$
3: $M \leftarrow (crand_{bc}||o_{bc}||Payload||verDep||stateUpdate||retryFlag)$
4: $(\mathsf{pk}, \mathsf{sk}) \overset{r}{\leftarrow} \mathsf{KeyGen}(1^k)$
5: $\sigma \leftarrow \mathsf{BlindSign}((\mathcal{U}(M, \mathsf{pk}), (\mathcal{A}(\mathsf{sk}))$
6: $trans_{prop} \leftarrow (crand_{bc}, o_{bc}, Payload, stateUpdate,\ verDep, retryFlag, \sigma)$

7: return $< propose, trans_{prop} >$
```

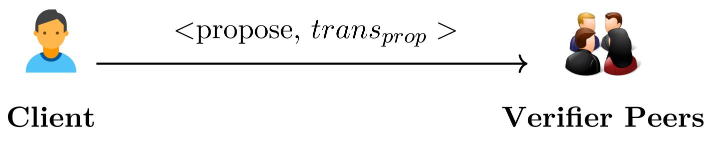


Figure 3.4 – Transaction Proposal.

The next step corresponds to the **Transaction Validation** detailed in Algorithm 2. The verifier peer vp_{bc_i} starts the process validating the blinded signature σ using the function VerifyBlindSign for the message $M = (crand||o_{bc}||Payload||verDep||stateUpdate\ ||retryFlag)$, and the public key pk. If the verifier gets a valid response; he continues validating the *verDep*, *stateUpdate*, and the *securityPolicies*. If *verDep* corresponds to the last version that the verifier peer has locally, and the *stateUpdate* is the same that its local state and the *securityPolicies* authorizes the instruction o_{bc}, the peer validates the transaction as it is described in Figure 3.5). Otherwise, he rejects as it is presented in Figure 3.6. For the transaction to be validated, the verifier vp_{bc_i} uses the function BlindSign to sign the message *transaction-valid* with his corresponding *tid*. Finally, once the transaction is validated, the verifier peer vp_{bc_i} creates the *transaction-valid* message to be sent to the client c_{bc}.

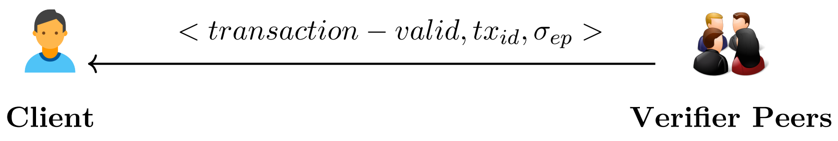


Figure 3.5 – Transaction Validation: Valid Transaction.

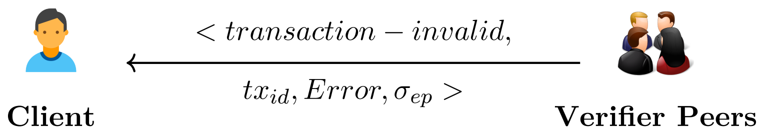


Figure 3.6 – Transaction Validation: Invalid Transaction.

The next step is the **Broadcasting to Consensus**. The client c_{bc} then receives the responses from the verifier peers. These responses are collected inside of an array called *validations*. Once the client has collected enough valid responses (σ_{ver_i}) in the *validations* array (at least $50\% + 1$), the client sends the transaction to the consensors to be included in the next block by using the function *broadcast* (see Algorithm 3 and Figure 3.7 for a detailed presentation).

Algorithm 2 TxEndors($trans_{prop}$, $securityPolicies$, pk)

```
1: $tid \leftarrow trans_{prop}.Payload.tid$
2: $M \leftarrow (trans_{prop}.crand_{bc}||trans_{prop}.o_{bc}||trans_{prop}.Payload$
   $||trans_{prop}.verDep||trans_{prop}.stateUpdate||trans_{prop}.retryFlag)$
3: $\sigma \leftarrow trans_{prop}.\sigma$
4: $(\mathsf{pk}_{ver_i}, \mathsf{sk}_{ver_i}) \leftarrow \mathsf{KeyGen}(1^k)$
5: if $\mathsf{VerifyBlindSign}(M, \sigma, \mathsf{pk})$ $==$ $invalid$ then
6:    $result \leftarrow$ (transaction-invalid$||tid||$incorrect-signature)
7:    $\sigma_{ver_i} \leftarrow \mathsf{BlindSign}((\mathcal{U}(result, \mathsf{pk}_{ver_i}), (\mathcal{A}(\mathsf{sk}_{ver_i}))$
8:    return <transaction-invalid,tid,incorrect-signature, $\sigma_{ver_i}$ >
9: else
10:   $(verDep_{ver_i}, stateUpdate_{ver_i}) \leftarrow$
      $EXEC(trans_{prop}.o_{bc}, trans_{prop}.Payload)$
11:   if $trans_{prop}.stateUpdate \neq stateUpdate_{ver_i}$ then
12:      $result \leftarrow$ (transaction-invalid$||tid||$incorrect-state)
13:      $\sigma_{ver_i} \leftarrow \mathsf{BlindSign}((\mathcal{U}(result, \mathsf{pk}_{ver_i}), (\mathcal{A}(\mathsf{sk}_{ver_i}))$
14:      return <transaction-invalid,tid,incorrect-state, $\sigma_{ver_i}$ >
15:   else if $trans_{prop}.varDep \neq varDep_{ver_i}$ then
16:      $result \leftarrow$ (transaction-invalid$||tid||$incorrect-version)
17:      $\sigma_{ver_i} \leftarrow \mathsf{BlindSign}((\mathcal{U}(result, \mathsf{pk}_{ver_i}), (\mathcal{A}(\mathsf{sk}_{ver_i}))$
18:      return <transaction-invalid,tid,incorrect-version, $\sigma_{ver_i}$ >
19:   else if $securityPolicies == invalid$ then
20:      $result \leftarrow$ (transaction-invalid$||tid||$rejected)
21:      $\sigma_{ver_i} \leftarrow \mathsf{BlindSign}((\mathcal{U}(result, \mathsf{pk}_{ver_i}), (\mathcal{A}(\mathsf{sk}_{ver_i}))$
22:      return <transaction-invalid,tid,rejected, $\sigma_{ver_i}$ >
23:   else
24:      $result \leftarrow (transaction\text{-}valid||tid)$
25:      $\sigma_{ver_i} \leftarrow \mathsf{BlindSign}((\mathcal{U}(result, \mathsf{pk}_{ver_i}), (\mathcal{A}(\mathsf{sk}_{ver_i}))$
26:      return $<$ transaction-valid, $tid, \sigma_{ver_i} >$
27:   end if
28: end if
```

Algorithm 3 TxBrodCons($trans_{prop}$, σ_{ver_i}, $validations_{tid}$)

```
1: $validations_{tid}[validations_{tid}.length + 1] \leftarrow \sigma_{ver_i}$
2: if $(validations_{tid}.length \geq (\frac{TotVerPeers}{2} + 1))$ then
3:    $blob \leftarrow (trans_{prop}, validations_{tid})$
4:    return $< consensus, blob >$
5: end if
```

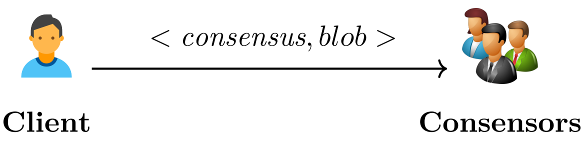


Figure 3.7 – Broadcasting to consensus.

Finally, during the **Commitment**, described in Algorithm 6, the transaction is validated with its respective validations. If the *verDep* has not changed during the validation process, and the *validations* are valid according to the security policies for the operation o_{bc}; the transaction is added into the new block. The *NewBlock* is an array where we store the new transactions to be settled; and it is proposed by the consensor leader. Once we have reached the maximum block size (*BlockLengthMax*) configured in the network, the consensor leader spreads the new block in a $propose_{block}$ message to the consensors backups according to the commitment process described in Algorithm 4.

The backups, denoted by $consp_{bc_i}$ for $i = 1, \ldots, k-1$, where k is the total number of backups, validate the $propose_{block}$ message. Then they respond to the rest of the consensors with a $prepare_{block}$ message as explained in Algorithm 5.

Once each of the consensor receives enough valid $prepare_{block}$ messages, it means at least $50\% + 1$ of the total of the consensors, it notifies to the nodes connected to the network that the validation process has finished successfully, and he commits the block in their local version of the blockchain by sending a $commit_{block}$ message as described in Algorithm 6 and in Figure 3.8. In the case that the validation process fails, the transactions in the block are rejected, and the *retryFlag* for those transactions are set to 1.

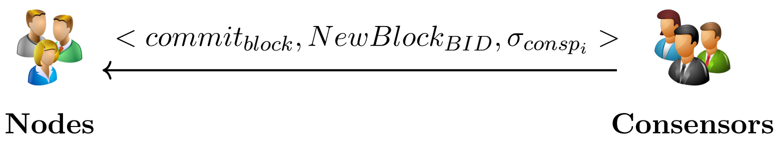


Figure 3.8 – Commit Block.

3.5 Protocol Properties

We show that our blockchain BlindCons is consistent and ensures liveness.

Algorithm 4 TxBrodCons($blob, SecurityPolicies$)

```
1: $M \leftarrow (blob.trans_{prop}.crand_{bc}||blob.trans_{prop}.o_{bc}|| blob.trans_{prop}.Payload\ ||blob.trans_{prop}.verDep\ ||blob.trans_{prop}.stateUpdate\ ||blob.trans_{prop}.retryFlag)$
2: $\sigma \leftarrow blob.trans_{prop}.\sigma$
3: $(\mathsf{pk}_{consp_{leader}}, \mathsf{sk}_{consp_{leader}}) \leftarrow \mathsf{KeyGen}(1^k)$
4: if $\mathsf{VerifyBlindSign}(M, \sigma, \mathsf{pk})$ == invalid then
5:     $result \leftarrow (prepare_{block}||NewBlock_{BID})||incorrect\text{-}signature)$
6:     $\sigma_{consp_{leader}} \leftarrow \mathsf{BlindSign}((\mathcal{U}(result, \mathsf{pk}_{consp_{leader}})\ (\mathcal{A}(\mathsf{sk}_{consp_{leader}}))$
7:     return $<transaction\text{-}invalid, tid, incorrect\text{-}signature,\ \sigma_{consp_{leader}}>$
8: else
9:     $BID \xleftarrow{r} \mathbb{N}$
10:    if $(securityPolicies == invalid)||\ (blob.trans_{prop}.VerDep \neq valid)$ then
11:        $result \leftarrow (transaction\text{-}invalid||tid||invalid\text{-}sec\text{-}conditions)$
12:        $\sigma_{consp_{leader}} \leftarrow \mathsf{BlindSign}((\mathcal{U}(result, \mathsf{pk}_{consp_{leader}}), (\mathcal{A}(\mathsf{sk}_{consp_{leader}}))$
13:        return $<transaction\text{-}invalid, tid, invalid\text{-}sec\text{-}conditions, \sigma_{consp_{leader}}>$
14:    else
15:        $NewBlock_{BID}[NewBlock_{BID}.length + 1] \leftarrow blob$
16:    else if $NewBlock_{BID}.length == BlockLengthMax$ then
17:        $result \leftarrow (propose_{block}||NewBlock_{BID})$
18:        $\sigma_{consp_{leader}} \leftarrow \mathsf{BlindSign}((\mathcal{U}(result, \mathsf{pk}_{consp_{leader}}), (\mathcal{A}(\mathsf{sk}_{consp_{leader}}))$
19:        return $<propose_{block}, NewBlock_{BID}, \sigma_{consp_{leader}}>$
20:    else
21:        return wait-for-next-transaction
22:    end if
23: end if
```

3.5.1 Consistency

A protocol is said to be *consistent* if it ensures that a transaction generated by a valid user stays immutable in the blockchain.

Definition 3.5. *A protocol $\mathbb{P}$ is $T-$consistent if a transaction tx generated by an honest client c_{cb} to execute a valid operation o_{bc}, it is confirmed and stays immutable in the blockchain after $T-$ round of new blocks.*

Theorem 3.1. *BlindCons protocol is 1-consistent.*

Proof. The protocol described in Section 3.4 is a BFT based consensus algorithm. Consistency is achieved by agreeing with the validity of the transaction through a Byzantine Agreement process. Hence, for a transaction tx

Algorithm 5 Prepare($NewBlock_{BID}, \sigma_{consp_{leader}}, SecurityPolicies$)

```
1:  $(\mathsf{pk}_{consp_{bc_i}}, \mathsf{sk}_{consp_{bc_i}}) \leftarrow \mathsf{KeyGen}(1^k)$
2:  $M \leftarrow (propose_{block} || NewBlock_{BID})$
3:  if $\mathsf{VerifyBlindSign}(M, \sigma_{consp_{leader}}, \mathsf{pk}_{consp_{leader}}) == invalid$ then
4:      $result \leftarrow (block\text{-}invalid || NewBlock_{BID}) || incorrect\text{-}signature)$
5:      $\sigma_{consp_{bc_i}} \leftarrow \mathsf{BlindSign}((\mathcal{U}(result, \mathsf{pk}_{consp_{bc_i}}), (\mathcal{A}(\mathsf{sk}_{consp_{bc_i}}))$
6:      return $< block - invalid, NewBlock_{BID},$
        $incorrect - signature, \sigma_{consp_{bc_i}} >$
7:  else
8:      for all $i = 0$ to $BlockLengthMax$ do
9:          if $(SecurityPolicies == invalid)||$
            $(NewBlock_{BID}[i].trans_{prop}.verDep \neq valid)$ then
10:             $NewBlock_{BID}[i] \leftarrow (NewBlock_{BID}[i].trans_{prop},$
                $NewBlock_{BID}[i].validations_{tid}, invalid)$
11:             $invalid_{count} \leftarrow invalid_{count} + 1$
12:         end if
13:     end for
14:     if $invalid_{count} \neq 0$ then
15:         $result \leftarrow (block\text{-}invalid || NewBlock_{BID} || invalid\text{-}transaction)$
16:         $\sigma_{consp_{bc_i}} \leftarrow \mathsf{BlindSign}((\mathcal{U}(result, \mathsf{pk}_{consp_{bc_i}}), (\mathcal{A}(\mathsf{sk}_{consp_{bc_i}}))$
17:         return $< prepare_{block}, NewBlock_{BID},$
            $invalid - transaction, \sigma_{consp_{bc_i}} >$
18:     else
19:         $result \leftarrow (prepare_{block} || NewBlock_{BID})$
20:         $\sigma_{consp_{bc_i}} \leftarrow \mathsf{BlindSign}((\mathcal{U}(result, \mathsf{pk}_{consp_{bc_i}}), (\mathcal{A}(\mathsf{sk}_{consp_{bc_i}}))$
21:         return $< prepare_{block}, NewBlock_{BID}, \sigma_{consp_{bc_i}} >$
22:     end if
23: end if
```

that has reached a majority of valid endorsements for an operation o_{bc}, the probability of not settling it in a new block and having forks in the chain is neglected if we have at most $\lfloor \frac{n-1}{3} \rfloor$ out of total n malicious peers, as it has been shown in [CL99, LM07] under the terminology of safeness. It is 1-consistent because we do not have any fork; hence only one block is needed to wait to have a transaction validated. □

3.5.2 Liveness

The liveness property means that a consensus protocol ensures that if an honest client submits a valid transaction, a new block will be appended to the chain with the transaction in it. Hence, the protocol must ensure that

Algorithm 6 BlockCom($NewBlock_{BID}, securityPolicies, retryFlag$)

```
1: $M \leftarrow (prepare_{block}||NewBlock_{BID})$
2: if VerifyBlindSign($M, \sigma_{consp_{bc_i}}, \mathsf{pk}_{consp_{bc_i}}$) $== invalid$ then
3:     $result \leftarrow (block\text{-}invalid||NewBlock_{BID})||incorrect\text{-}signature)$
4:     $\sigma_{consp_{bc_i}} \leftarrow$BlindSign($(\mathcal{U}(result, \mathsf{pk}_{consp_{bc_i}}), (\mathcal{A}(\mathsf{sk}_{consp_{bc_i}}))$)
5:     return $< block - invalid, NewBlock_{BID},$
       $incorrect - signature, \sigma_{consp_{bc_i}} >$
6: else
7:     $count_{block_{BID}} \leftarrow count_{block_{BID}} + 1$
8:     if $count_{block_{BID}} < CountMinForNewBlock$ then
9:         return WAIT-FOR-NEXT-BLOCK
10:    else
11:        $result \leftarrow (commit_{block}||NewBlock_{BID})$
12:        $\sigma_{consp_{bc_i}} \leftarrow$ BlindSign($(\mathcal{U}(result, \mathsf{pk}_{consp_{bc_i}}), (\mathcal{A}(\mathsf{sk}_{consp_{bc_i}}))$)
13:        return $< commit_{block}, NewBlock_{BID}, \sigma_{consp_i} >$
14:    end if
15: end if
```

the blockchain grows if valid clients generate valid transactions.

Definition 3.6 (Liveness). *A consensus protocol $\mathbb{P}$ ensures* liveness for a blockchain C *if* $\mathbb{P}$ *ensures that after a period of time t, the new version of the blockchain C' is $C' > C$, if a valid client $c_{i_{bc}}$ has broadcasted a valid transaction tx_i during the time t.*

Theorem 3.2. *BlindCons achieves liveness.*

Proof. BlindCons is a BTF-based consensus protocol. Thus, liveness is achieved if after the transaction validation process, the ordering services propose a new block B with the transactions broadcasted by the clients during a period of time t. Hence, for valid transactions tx_i, where $i \in \mathbb{N}_0$, issued by valid a client $c_{i_{bc}}$ during a period of time t, the probability that $C' = C$ is neglected if we have at most $\lfloor \frac{n-1}{3} \rfloor$ out of total n malicious peers [CL99]. □

3.6 Conclusion

We propose a new consensus algorithm for privacy-preserving user identity using blind signatures scheme for transactions unlinkability. This algorithm is based on an *execute-order-validate* process proposed in Hyperledger Fabric. However, our construction uses a BFT algorithm to agree on the new block instead of Kafka or Raft. Our protocol allows the users to issue a blinded signature that keeps their identity private during the Blockchain transaction validation process and after the transaction is settled. The signing scheme is based on an abstract model that can be used with different blind signature constructions like Okamoto-Schnorr blind signature [Oka92], Chum's blind signature [Cha82], Fuchsbauer et al. [FHS15], among others. The protocol was designed to implement a blind signature scheme to achieve transactions unlinkability by using the components that any Permissioned or Private Blockchain architecture has. However, our construction uses a BFT algorithm to agree on the new block instead of Kafka or Raft. Hence, our algorithm performs the signature blinding process by using the membership authorities or the user administrators that already exist in the Permissioned scheme, making it efficient for these kinds of Blockchain architectures. Moreover, our protocol can be easily adapted in Permissioned Blockchains like Hyperledger Fabric or Iroha.

Chapter 4

A Privacy Preserving e-Voting Protocol for Permissioned Blockchain

Contents

With the immutability property and decentralised architecture, Blockchain technology is considered as a revolution for several topics. For example, electronic voting, it can be used to ensure voter privacy, the integrity of votes, and the verifiability of vote results. More precisely permissioned Blockchains could be the solution for many of the e-voting issues. In this work, we start by evaluating some of the existing Blockchain-based e-voting systems and analyse their drawbacks. We then propose a fully-decentralised e-voting system based on permissioned Blockchain. Called DABSTERS, our protocol uses a blinded signature consensus algorithm to preserve voter's privacy. This ensures several security properties and aims at achieving a balance between voter privacy and election transparency. Furthermore, we formally prove the security of our protocol by using the automated verification tool, ProVerif, with the Applied Pi-Calculus modelling language.

4.1 Introduction

Voting is the cornerstone of a democratic country. The list of security properties that must respect a secure voting protocol includes the following features. **Eligibility:** only registered voters can vote and only one vote per voter is counted. If the voter is allowed to vote more than once, the most recent ballot will be tallied and all others must be discarded. **Individual verifiability:** the voter him/herself must be able to verify that his/her ballot was cast as intended and counted as cast. **Universal verifiability:** after the tallying process, the results are published and must be verifiable by everybody. **Vote-privacy:** the connection between a voter and his/her vote must not be reconstructable without his/her help. **Receipt-freeness:** a voter cannot prove to a potential coercer that he/she voted in a particular way. **Coercion resistance:** even when a voter interacts with a coercer during the voting process, the coercer will be not sure whether the voter obeyed their demand or not. **Integrity:** ballots are not altered or deleted during any step of the election. **Fairness:** no partial results are published before tallying has ended; otherwise voters may be influenced by these results and vote differently. **Robustness:** the system should be able to tolerate some faulty votes. **Vote-and-go:** a voter does not need to wait for the end of the voting phase or trigger the tallying phase. **Voting policy:** specify if a voter has the right to vote more than once or he/she has not the right to change his/her opinion once he/she cast a vote.

Traditionally, during an election, the voter goes to a polling station and makes his/her choice in an anonymously, without any external influence. To perform the tally, we need to trust a central authority. From this comes the risk of electoral fraud. The tallying authority has the possibility to falsify votes and thus to elect a candidate who should not be elected. It is also possible for the registration authority to allow ineligible voters to vote. Hence, voting becomes useless and we notice a decrease in voter turnout in elections. Decentralised systems can be a good alternative to traditional voting since we need a secure, verifiable and privacy preserving e-voting systems for our elections. Blockchain is a distributed ledger that operates without the need to a trusted party. Expanding e-voting into Blockchain technology could be the solution to alleviate the present issues in voting.

Due to the proliferation of Blockchain implementations, the European Blockchain Observatory and Forum has published a technical report [14]

where it recommends the use of private or permissioned Blockchains for sensitive data storage, which is the architecture implemented in an e-voting system. In this Blockchain architecture, the user credentials are generated by a Certificate Authority (CA). Hence, the users must be enrolled into the system through the CA before joining the network. This model is suitable for an e-voting system because the user management can rely on the Blockchain platform, due to their formal enrolling process. The advantage of having a minimum level of trust through our knowing the participants is that we can achieve security for the Blockchain replication process by using Byzantine Agreement as a consensus mechanism. Although permissioned Blockchains have several features suitable for services that involve sensitive data, such as user's personal information, they have drawbacks related to transactions and user linkability. This is bceause each user credential, public key pair and certificates, are issued for specific users that were previously enrolled in the CA. In order to overcome this drawback, we use, in this work, the Okamoto-Schnorr blind signature scheme to sign the transactions without linking the user to it. This model allows validating transactions without exposing the user's identification, and therefore maintaining the votes's privacy.
Related Work: In the last few decades, many Blockchain-based e-voting protocols have been proposed to address the security issues of traditional voting protocols. Due to the limitation on the number of pages, we give a brief overview of some of these systems and evaluate their security in Table 4.1, in which we use the following abbreviations[1].

- *Open Vote Network (OVN)* [MSH17]: It is a self-tallying, boardroom scale e-voting protocol implemented as a smart contract in Ethereum. This protocol guarantees voter's privacy and removes the need to trust the tallying authorities whether to ensure the anonymity of voters or to guarantee the verifiability of elections. However, it suffers from several security issues. For example, it supports only elections with two options (yes or no) and with a maximum of 50 voters due to the mathematical tools used and to the gas limit for blocks imposed by Ethereum. Additionally, this protocol does not provide any mechanism to ensure coercion resistance and must trust the election administrator to ensure voters' eligibility. Open Vote Network is not resistant to the misbehavior of a dishonest miner who can invalidate the election by modifying voters' transactions before storing them on blocks. A dishonest voter can also invalidate the election by sending an invalid vote or by abstaining during the voting phase.

- *E-Voting with Blockchain: An E-Voting Protocol with Decentralization and Voter Privacy (EVPDVP)* [HAM18]: Implemented on a private network that uses the Ethereum Blockchain API, this protocol

[1]TCA: Trusted Central Authority; SV: Single Vote; MV: Multiple Votes.

uses the blind signature to ensure voters privacy. It needs a central authority (CA) as a trusted party to ensure voters eligibility and allow voters to change and update their votes. To ensure fairness, voters include in their ballots a digital commitment of their choices instead of the real identity of the chosen candidate. To tally ballots, voters must broadcast to the network a ballot opening message during the counting phase.

- *Verify-Your-Vote: A Verifiable Blockchain-based Online Voting Protocol (VYV)* [CYLR18]: An online e-voting protocol that uses Ethereum Blockchain as a bulletin board. It is based on a variety of cryptographic primitives, namely Elliptic Curve Cryptography [HM05], pairings [Bon12, RS15] and Identity Based Encryption [BF03]. The combination of security properties in this protocol has numerous advantages. First, it ensures voter's privacy because the Blockchain is characterized by the anonymity of its transactions. It also ensures fairness, individual and universal verifiability because the ballot structure includes counter-values, which serve as receipts for voters, and homomorphism of pairings. However, the registration phase of this protocol is centralised. A unique authority, which is the registration agent, is responsible for verifying the eligibility of voters and registering them. A second problem is inherent in the use of Ethereum because each transaction sent by the protocol entities in the Blockchain passes through miners who validate it, put it in the current block and execute the consensus algorithm. Therefore, any dishonest miner in the election Blockchain can modify transactions before storing them on blocks. Additionally, this protocol is not coercion resistant.

- *TIVI* [Sma16]: It is a commercial online voting solution based on biometric authentication, designed by the company Smartmatic. It checks the elector's identity via a selfie using facial recognition technology. TIVI ensures the secrecy of votes so long as the encryption remains uncompromised. It also provides voters' privacy thanks to its mixing phase. It offers the possibility to follow votes by the mean of a QR code stored during the voting phase and checked later via a smartphone application. However, this system does not provide any mechanism to protect voters from coercion or to ensure receipt-freeness. Additionally, TIVI uses the Ethereum Blockchain as a ballot box. Hence, it is not resistant to misbehaving miners that could invalidate the election by modifying votes before storing them on the election Blockchain.

- *Follow My Vote (FMV)* [Fol12]: It is a commercial online voting protocol that uses the Ethereum Blockchain as a ballot box. A trusted authority authenticates eligible voters and provides them with passphrases needed in case of changing their votes in the future. Voters can watch the election progress in real-time as votes are cast. It includes an authentication phase that ensures voters' eligibility. In addition, it allows voters to locate their votes, and check that they are both present and correct using their voters' IDs. Nevertheless, this voting system requires a trusted authority to ensure votes confidentiality and hide the correspondence between the voters' real identities and their voting keys. If this authority is corrupted, votes are no longer anonymous. This system does not verify votes secrecy because votes are cast without being encrypted. Moreover, the ability to change votes, coupled with the ability to observe the election in real-time compromise fairness property. This system is not coercion resistance and is not universally verifiable because, we have no way to verify that the votes present in the election final election result are cast by eligible voters.

- *BitCongress [Inc]*: A commercial online voting platform based on three networks: Bitcoin, Counterparty (a decentralised asset creation system and decentralised asset exchange) and a Smart Contract Blockchain. It aims at preventing double voting by using the time stamp system of the Bitcoin Blockchain. This platform does not ensure voters eligibility because it allows any Bitcoin address to register for the election. It performs the tally using, by default, a modified version of Borda count and a Quota Borda system for large scale elections. It ensures individual and universal verifiability but it is not coercion resistant.

- *Platform-independent Secure Blockchain-based Voting System (PSBVS) [YLS+18]:* Implemented in the Hyperledger Fabric Blockchain [ABB+18], this protocol uses Paillier cryptosystem [Pai11] to encrypt votes before being cast, proof of knowledge to ensure the correctness and consistency of votes, and Short Linkable Ring Signature (SLRS) [ACST06] to guarantee voters privacy. On the other hand, this protocol does not include a registration phase in which we verify, physically or by using biometric techniques, the voter's eligibility. A voter can register him/herself by simply providing his/her e-mail address, identity number or an invitation URL with the desired password. However, these mechanisms are not sufficient to verify a voter's eligebility and information like e-mail address or identity number can be known by people other than the voter him/herself. Also, concerning the definition of coercion resistance given by Juels et al. [JCJ10], this protocol is not coercion resistant. If a voter gives his/her secret key to a coercer, the coercer can vote in the place of the voter who cannot modify this vote later. We mention here that the coerced voter cannot provide a fake secret key to the coercer because the smart contract rejects a vote with a fake secret key

	OVN	**EVPDVP**	**VYV**	**TIVI**	**FMV**	**BitCongress**	**PSBVS**	**DABSTERS**
Eligibility	TCA	TCA	TCA	✓	TCA	X	X	✓
Individual verif	✓	✓	✓	✓	✓	✓	✓	✓
Universal verif	✓	✓	✓	X	X	✓	✓	✓
Vote-Privacy	✓	✓	✓	✓	TCA	✓	✓	✓
Receipt-freeness	X	✓	✓	X	X	X	✓	✓
Coercion resistance	X	X	X	X	X	X	X	X
Fairness	X	✓	✓	✓	X	X	✓	✓
Integrity	✓	✓	X	✓	X	✓	✓	✓
Robustness	X	✓	✓	✓	✓	✓	✓	✓
Vote-and-go	X	X	✓	✓	✓	✓	✓	✓
Voting policy	SV	MV	MV	SV	MV	MV	SV	MV

Table 4.1 – Security evaluation of OVN, EVPDVP, VYV, TIVI, FMV, BitCongress, PSBVS and DABSTERS.

Contributions: In this work, we aim at designing a secure online e-voting protocol that addresses the security issues mentioned in the related work section by using the Blockchain technology and a variety of cryptographic primitives. Called DABSTER, our protocol uses a new architecture based on permissioned Blockchain and blind signature. It satisfies the following security properties: eligibility, individual verifiability, universal verifiability, vote-privacy, receipt-freeness, fairness, integrity and robustness. Our contributions can be summarised as follows:

- A new architecture of trust for electronic voting systems. This architecture is based on permissioned Blockchain and on a blind consensus which provides voter's privacy and vote's integrity.
- A secure and fully distributed electronic voting protocol based on our propounded architecture.
- A detailed security evaluation of the protocol and a formal security proof using the Applied Pi-Calculus modelling language and the automated verification tool ProVerif.

Outline: In the next section, we give an overview of the Byzantine Fault Tolerance (BFT) with blind signature consensus algorithms. Then in Section 4.3, we describe our proposed e-voting protocol, DABSTERS, and give its different stakeholders and phases, and the structure of each voter's ballot. Finally, in Section 4.4, we evaluate our protocol's security using Proverif when possible. The conclusion is a summary of DABSTERS and a proposal for ongoing evaluation of its performance.

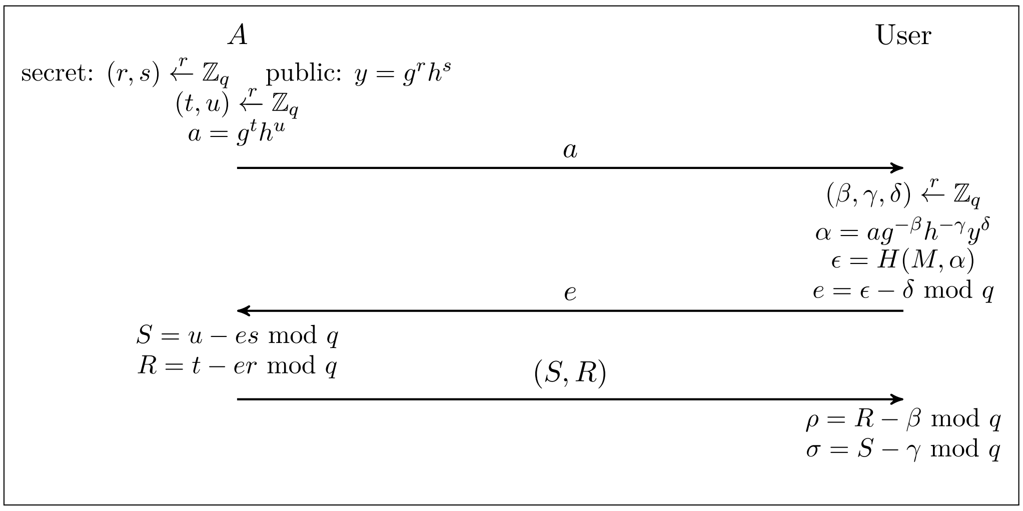

A
User
secret: $(r,s) \overset{r}{\leftarrow} \mathbb{Z}_q$ public: $y = g^r h^s$
$(t,u) \overset{r}{\leftarrow} \mathbb{Z}_q$
$a = g^t h^u$
a
$(\beta,\gamma,\delta) \overset{r}{\leftarrow} \mathbb{Z}_q$
$\alpha = a g^{-\beta} h^{-\gamma} y^{\delta}$
$\epsilon = H(M,\alpha)$
$e = \epsilon - \delta \bmod q$
e
$S = u - es \bmod q$
$R = t - er \bmod q$
(S,R)
$\rho = R - \beta \bmod q$
$\sigma = S - \gamma \bmod q$

Figure 4.1 – Okamoto-Schnorr blind signature diagram, where $y \overset{r}{\leftarrow} \mathbb{Z}_q$ means that y is randomly chosen in $\mathbb{Z}_q$.

4.2 Background

We define the Okamoto-Schnorr blind signature, before using it in a Byzantine based consensus.

4.2.1 Blind Signature

Let p and q be two large primes with $q|p-1$. Let G be a cyclic group of prime order q, and g and h be generators of G. Let $H : \{0,1\}^* \to \mathbb{Z}_q$ be a cryptographic hash function.

Key Generation: Let $(r,s) \overset{r}{\leftarrow} \mathbb{Z}_q$ and $y = g^r h^s$ be the A's private and public key, respectively.

Blind signature protocol:

1. A chooses $(t,u) \overset{r}{\leftarrow} \mathbb{Z}_q$, computes $a = g^t h^u$, and sends a to the user.
2. The user chooses $(\beta,\gamma,\delta) \overset{r}{\leftarrow} \mathbb{Z}_q$ and computes the blind version of a as $\alpha = a g^{-\beta} h^{-\gamma} y^{\delta}$, and $\epsilon = H(M,\alpha)$. Then calculates $e = \epsilon - \delta \bmod q$, and sends e to the A.
3. A computes $S = u - es \bmod q$ and $R = t - er \bmod q$, sends (S,R) to the user.
4. The user calculates $\rho = R - \beta \bmod q$ and $\sigma = S - \gamma \bmod q$.

Verification: Given a message $M \in \{0,1\}^*$ and a signature (ρ,σ,ϵ), we have $\alpha = g^{\rho} h^{\sigma} y^{\epsilon} \bmod p$.

The Okamoto-Schnorr blind signature scheme is suitable with a private Blockchain architecture due to the blinding process that can be performed by the same authority responsible of the enrollment process (see Figure 4.1, where the authority A blindly signs a message for the user). We use *BlindSign(M,*$(\beta,\sigma,\gamma),y$*)* and *VerifyBlindSign(M,*$(\rho,\delta,\epsilon),y$*)* to blind sign and to verify the blinded signature, respectively using Okamoto-Schnorr, where M corresponds to the message to be signed, (β,σ,γ) to the secret values randomly chosen, (ρ,δ,ϵ) to the blinded signature; and y to the RA's public key. The result obtained from the unction *BlindSign* corresponds to the blinded signature (ρ,σ,ϵ). On the other hand, the function *VerifyBlindSign* returns a response *valid* or *invalid*.

4.2.2 BFT based consensus algorithm

Considering a permissioned Byzantine Fault Tolerance (BFT) based consensus protocol like the one introduced in Hyperledger Fabric [ABB+18]. In this protocol, the digital signature is used as a user authentication method without protecting the user privacy. Hence, for a privacy-preserving consensus protocol, we need to add the following properties to the BFT based consensus algorithm:

- Alice sends a newly signed transaction to the registration authority (RA) which is responsible for the enrollment of Alice.
- Alice's signature is validated only by the RA.
- The RA anonymises Alice's identity.
- The RA signs the transaction sent by Alice to the network.
- All the node in the transactions validation process can validate the RA's signature.
- The RA signature cannot be duplicated.

Now, to keep the client's and peers' privacy involved in the transactional process, we need to hide his ID and make his signature blind. However, we do not address the ID hiding process with any particular mechanism. Therefore, we consider that the ID is replaced by a value corresponding to the anonymised user ID, and this process can be performed by using different schemes. As presented in [KLMN18], to address the issue related to the digital signature, we replace the signing mechanism used in the original protocol with the Okamoto-Schnorr blind signature scheme [Oka92]. In order to maintain the consistency and liveness that the protocol has, we keep the transactional flow. However, the steps are modified to accept the new blind signature scheme to authenticate the clients and peers.

The transactional process based on our BFT consensus algorithm with Blind Signature consists of the following steps:

1. **Initiating Transactions:** The client c_{bc} generates a message M to execute an operation o_{bc} in the network with a blinded signature by using $\mathit{BlindSign}((M, \beta, \sigma, \gamma), y)$.

2. **Transaction Proposal:** The submitting peer sp_{bc} receives the message M coming from the client c_{bc}, validates the client blinded signature by using *VerifyBlindSign(M,*$(\rho, \delta, \epsilon), y)$ and proposes a transaction with the client instruction o_{bc}.

3. **Transaction Endorsement:** The endorser peers ep_{bc} validate the client blinded signature using *VerifyBlindSign(M,*$(\rho, \delta, \epsilon), y)$ and verify if the transaction is valid by simulating the operation o_{bc} using his local version of the Blockchain. Then, the endorser peers generate signed transactions with the result of the validation process and send it to the submitting peer sp_{bc}.

4. **Broadcasting to Consensus:** The submitting peer sp_{bc} collects the endorsement coming from the endorsing peers connected to the network. Once sp_{bc} collects enough valid answers from the endorsing peers, it broadcasts the transaction proposal with the endorsements to the ordering service.

5. **Commitment:** All the transactions are ordered within a block, and are validated with their respective endorsement. Then, the new block is spread through the network to be committed by the peers.

4.3 Description of DABSTERS

Our protocol is implemented over a new architecture of trust. It is based on a BFT-based consensus protocol [ABB$^+$18] and on a blinded signature consensus protocol, called *BlindCons* [KLMN18], presented in Section 4.2. It eliminates the risk of invalidating the election because of dishonest miners who modify the transactions before storing them on blocks. We also propose a distributed enrollment phase to reduce the need to trust election agents and impose the publication of the list of eligible registered voters at the end of the enrollment phase. This list is auditable and verifiable by all parties.

Our scheme unfolds in 5 stages. It starts with an enrollment phase in which registration authorities (RAs) verify the eligibility of voters by verifying the existence of their names and their identity card numbers in a list published beforehand and containing the names of all persons who have the right to vote. Then, all eligible voters are registered and provided with credentials. The enrollment phase is offline. At the end of this phase, RAs construct a list containing the names of all registered eligible voters and their ID card numbers. This list can be rejected or published on the election Blockchain during the validation phase. Once the list is validated, we move to the third stage which is the voting phase. Each eligible voter (V_i) initiates a transaction in which he/she writes his/her encrypted vote, signs the transaction using his/her credential and sends it to the RAs to check his/her signature and blind it. Then, the voter sends the transaction with the blinded signature and his/her anonymous ID (his/her credential) to the consensus peers to be validated and stored in the election Blockchain anonymously. After validating and storing all votes in our Blockchain, tallying authorities (TAs) read these encrypted votes from the network, decrypt them, and proceed to the tally. The final stage is the verification phase. During this phase, voters make sure that their votes have been considered correctly and check the accuracy of the tally. The individual verifiability is ensured due to the structure of our ballots and the universal verification is ensured thanks to the homomorphism property of pairings. Except for the enrollment phase, all the phases of our protocol are on-chain. Therefore, we call the BFT based consensus protocol with each transaction initiated by authorities and the BlindCons with each transaction initiated by eligible voters because we do not need to hide the identity of our authorities. Still, we need to ensure voter's privacy. In the following, we give a detailed description of the role of our protocol stakeholders, the structure of our ballot, the different protocol phases and the two consensus.

4.3.1 Protocol Stakeholders

DABSTERS involves three main entities:

- *Registration authorities (RAs)*: they verify the eligibility of every person wishing to register for the election and provide eligible voters with their credentials which are constructed by cooperation between all RAs.
- *Eligible voters (V)*: every eligible voter (V_i) has the right to vote more than once before the end of the voting phase and only his/her last vote is counted. Voters can verify that their votes are cast as intended and counted as cast during the verification phase. Also, they can check the accuracy of the final election result, but they are not obliged to participate in the verification phase (they can vote and go).
- *Tallying authorities (TAs)*: the protocol includes as many tallying authorities as candidates. Before the voting phase, they construct n ballots, where n is the number of registered voters. Thus, every voter has a unique ballot that is different from the other ballots. TAs encrypt ballots and send them to voters during the voting phase. They decrypt votes, calculate the final election result during the tallying phase, and publish the different values that allow voters to check the count's accuracy during the verification phase.

DABSTERS also involves observers and election organisers who have the right to host the Blockchain peers to ensure the correctness of the execution of the protocol.

4.3.2 Ballot Structure:

As illustrated in Figure 4.2, each ballot comprises a unique bulletin number BN calculated as follows: $BN = \{g, D\}_{PK_A}$, where g is a generator of an additive cyclic group G, D is a random number and PK_A is the administrator's public key. It also contains a set of candidates' names $name_j$ and candidates' pseudo IDs, denoted C_j, which are the candidates's position in the ballot, calculated from an initial order and an offset value. In addition, each ballot includes a set of counter-values $CV_{BN,name_j,k}$ that are receipts for each voter. They are calculated using the following formula: $CV_{BN,name_j,k} = e(Q_{namej}, S_k \cdot Q_{BN})$; where $e(.,.)$ is the pairing function, S_k is the secret key of the tallying authority TA_k, $Q_{namej} = H_1(name_j)$ and $Q_{BN} = H_1(BN)$ are two points of the elliptic curve E.

Ballot number BN			
Pseudo "ID C_j"	**Candidate's** "name $name_j$"	**Choice**	**Counter-value** "$CV_{BN,name_j,k}$"
0	Paul	☐	$CV_{BN,name_0,0}$
1	Nico	☐	$CV_{BN,name_1,1}$
2	Joel	☐	$CV_{BN,name_2,2}$

Table 4.2 – Ballot structure[PUT07].

4.3.3 Protocol Stages

Our solution includes the following phases:

4.3.3.1 Enrollment Phase:

Every person who has the right to vote and desires to do so, physically goes to the nearest registration station. He/she provides his/her national identity card to the RAs, who verify his/her eligibility by checking if his/her name and ID card number exist in a list, previously published, contains all persons that can participate in the election. If he/she is an eligible voter, the RAs save the number of his/her ID card and provide him with a credential that allows him to participate in the voting process. Voters' credentials are calculated using elliptic curve cryptography and have this form:

$Credential_{V_i} = S_M \cdot H_1(ID_{V_i})$ where:

- $S_M = S_1 \cdot S_2 \ldots, S_R$ is a secret master key calculated by cooperation between all RAs. Each registration authority participates with its secret fragment S_r; $r \in \{1 \ldots R\}$,
- H_1 is an hash function defined as follows: $H_1 : \{0,1\}^* \rightarrow G_1$; G_1 an additive cyclic group of order prime number q,
- ID_{V_i} is the number of the ID card of the voter V_i.

4.3.3.2 Validation Phase:

After registering all eligible voters, RAs create a list containing all registered voter's names and identity card numbers. This list should be viewable and verifiable by voters, election organisers and observers. Thus, RAs send this list in a transaction on our election Blockchain. This transaction passes through the five steps of the BFT based consensus protocol to be validated if the list is correct or rejected if the list contains names of ineligible voters.

- **Step1: Transaction initiation.** RAs generate the list of eligible voters to be validated by the network. Then, the list is sent to a submitter peer. In the case of an offline or misbehaving submitter peer, RAs send the transaction to the next submitter peer. This step is illustrated in Figure 4.2.

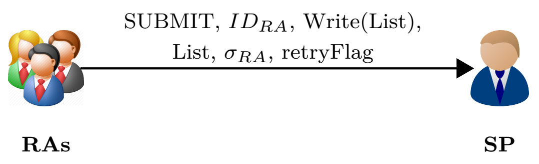

SUBMIT, ID_{RA}, Write(List),
List, σ_{RA}, retryFlag
RAs
SP

Figure 4.2 – Step1: Transaction initiation.

  - ID_{RA} is the ID of the registration authorities,
  - $Write(List)$ is the operation invoked by the RAs to be executed by the network. It consists of writing the list of eligible voters and their ID card numbers in the Blockchain,
  - *List* is the payload of the submitted transaction, which is the list of registered voters to be published on the Blockchain,
  - σ_{RA} is the signature of the registration authorities,
  - *retryFlag* is a boolean variable to identify whether to retry the transaction's submission in case of the transaction fails.

- **Step2: Transaction proposal.** The submitter peer receives the transaction and verifies the RAs signature. Then prepares a transaction proposal to be sent to the endorsing peers. Endorsing peers are composed of some voters, election organisers and observers who desire to host the Blockchain peers. This step is described in Figure 4.3.

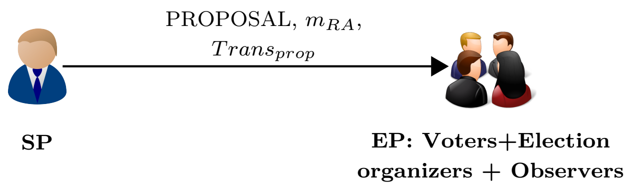


Figure 4.3 – Step2: Transaction proposal.

- $m_{RA} = (ID_{RA}, Write(List), List, \sigma_{RA})$
- $Trans_{prop} = (SP, Write(List), List, StateUpdate, VerDep)$:
  - *StateUpdate* corresponds to the state machine after simulating locally the operation coming in *Write(List)*.
  - *VerDep* is the version dependency associated with the variable to be created or modified. It is used to keep the consistency of the variables across the different machine state versions.

- **Step3: Transaction endorsement.** Each endorser peer verifies the signature of the registration authorities σ_{RA} coming in m_{RA} and checks that the list of eligible voters in m_{RA} and $Trans_{prop}$ is the same. Then, each endorser verifies the eligibility of all names and ID card numbers included in the list. If they are all valid, the endorser peer generates *a transaction valid message* to be sent to the submitter peer (Figure 4.4). But if the list includes names of ineligible voters, the endorser peer generates *a transaction invalid message* (Figure 4.5).

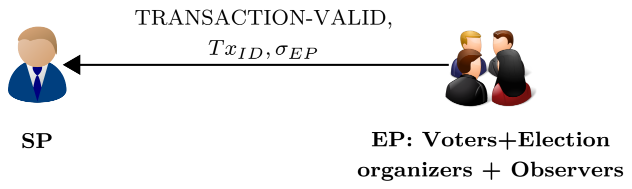


Figure 4.4 – Step3: Transaction endorsement: valid transaction.

- Tx_{ID} is the transaction ID,
- σ_{EP} is the signature of the endorser peer.
- *Error* message can have the following values:

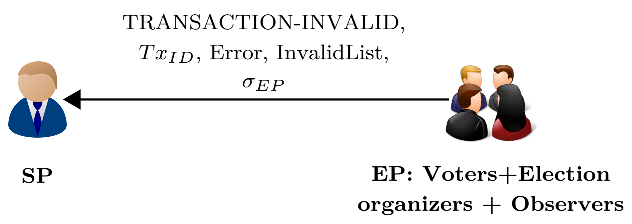

TRANSACTION-INVALID, Tx_{ID}, Error, InvalidList, σ_{EP}
SP
EP: Voters+Election organizers + Observers

Figure 4.5 – Step3: Transaction endorsement: invalid transaction.

    * INCORRECT-STATE: when the endorser tries to validate the transaction with a different local version of the Blockchain than the one coming in the transaction proposal.
    * INCORRECT-VERSION: when the version of the variable where the list will be recorded differs from the one referred in the transaction proposal.
    * REJECTED: for any other reason.
  - InvalidList: is the list of ineligible names that were included in the list sent by the RAs.

- **Step4: Broadcasting to Consensus.** The submitter peer waits for the response from the endorser peers. When it receives enough *Transaction Valid messages* adequately signed, the peer stores the endorsing signatures into packaged called endorsement. Once the transaction is considered endorsed, the peer invokes the consensus services by using *broadcast(blob)*, where *blob* = *(*$Trans_{prop}$*, endorsement)* (Figure4.6).

  The number of responses and endorsements to consider the transaction proposal as endorsed equals 50% + 1 of the total number of endorser peers. If the transaction has failed to collect enough endorsements, it abandons this transaction and notifies the RAs.

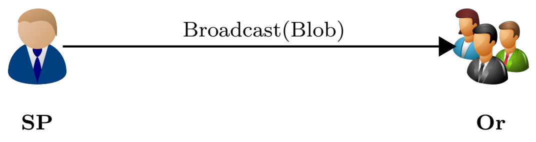


Figure 4.6 – Step4: Broadcasting to consensus.

- **Step5: Commitment.** Once the submitter peer broadcasts the transaction to consensus, the ordering services put it into the current block, which will be sent to all peers once built. Finally, if the transaction was not validated, the submitter peer SP informed the registration authorities.

In the case of an invalid list, the registration authorities have to correct the list and restart the validation phase. Thus, we move to the next phase (which is the voting phase) only when obtaining a valid list of registered voters.

4.3.3.3 Voting Phase:

Two entities participate during this phase:

- The tallying authorities who have constructed ballots before the beginning of the voting phase. To construct a ballot, TAs calculate, locally, the unique ballot number $BN = \{g, D\}_{PK_{TA}}$, the offset value $offset = H(g)\ mod\ m$ and the counter-values $CV_{BN,name_j,k} = e(Q_{namej}, S_k \cdot Q_{BN})$, where g is a generator of G an additive cyclic group of order a prime number, D is a random number, PK_{TA} is TAs' public key, m is the number of candidates, $e(.,.)$ is the pairing function, S_k is the secret key of the tallying authority TA_k, $Q_{namej} = H_1(name_j)$ and $Q_{BN} = H_1(BN)$ are two points of the elliptic curve E. Then, TAs choose, randomly, a blank ballot for each voter, encrypt it with the voter's public key and transmit it to the corresponding voter via the Blockchain. Ballots are sent encrypted because they contain secret information like the BN, the $offset$ and counter-values. To send encrypted ballots to voters via the Blockchain, TAs interact with the BFT consensus peers. These interactions unfold in five steps, the same steps as those presented in Section 4.3.3.2, and described in Figure 4.7.

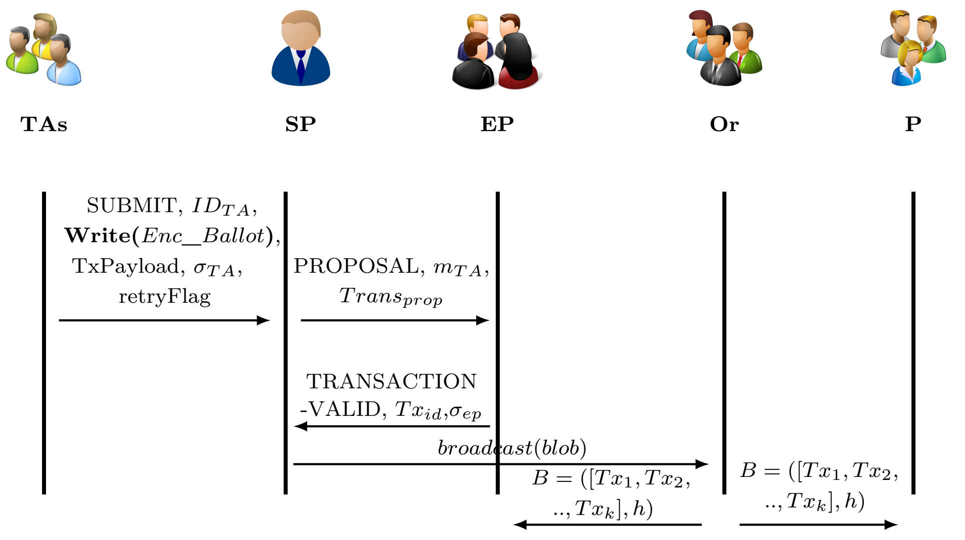

TAs
SP
EP
Or
P
SUBMIT, ID_{TA}, **Write**(Enc_Ballot), TxPayload, σ_{TA}, retryFlag
PROPOSAL, m_{TA}, $Trans_{prop}$
TRANSACTION -VALID, Tx_{id},σ_{ep}
$broadcast(blob)$
$B = ([Tx_1, Tx_2, .., Tx_k], h)$
$B = ([Tx_1, Tx_2, .., Tx_k], h)$

Figure 4.7 – Interaction between TAs and peers.

1. **Transaction initiation.** TAs initiate a transaction and send

it to a submitter peer SP. The transaction contains their ID (ID_{TA}), the list of encrypted ballots, the transaction payload, their signature (σ_{TA}) and the value of the variable retryFlag.

2. **Transaction proposal.** SP verifies the TAs signature and prepares a transaction proposal $Trans_{prop}$ = (SP, $Write(Enc\_Ballot), Enc\_Ballot, stateUpdate$, $VerDep$) to be sent to the endorsing peer with the TAs message $m_{TA} = (ID_{TA}$, $Write(Enc\_Ballot), Enc\_Ballot, \sigma_{TA})$
3. **Transaction endorsement.** EP verifies the σ_{TA} coming in m_{TA}, simulates the transaction proposal and validates that the $stateUpdate$ and $verDep$ are correct. If the validation process is successful, the endorser peer generates a transaction valid message to be sent to the submitter peer.
4. **Broadcasting to consensus.** When the SP receives a number of *Transaction Valid message* equals to 50% + 1 of the total number of endorser peers, adequately signed, he stores the endorsing signatures into an endorsement package and invokes the consensus services by using $broadcast(blob)$; where $blob = (Trans_{prop}, endorsement)$.
5. **Commitment.** Ordering services (Or) add the transaction to a block. Once they collect enough endorsed transactions, they broadcast the new block to all other peers. A block has the following form: $B = ([tx_1, tx_2, \ldots, tx_k]; h)$ where h corresponds to the hash value of the previous block.

- Every eligible voter retrieves his/her ballot, decrypts it using his/her secret key and encrypts his/her vote by voting then sends it in a transaction through the Blockchain. To encrypt his/her vote, the voter uses the Identity Based Encryption [BF03] and encrypts his/her ballot number BN with $Q_{C_j} = H_1(C_j)$ where C_j is the pseudo ID of the chosen candidate. Thus, each encrypted vote has the following form: $Enc\_Vote = \{BN\}_{Q_{C_j}}$.
  To be read from the Blockchain or be written on it, voters' transactions pass through the blinded signature consensus. We model in Figure 4.8 the steps through which a transaction of an eligible voter passes. First, we take the example of a transaction containing an encrypted vote.

  During the interactions between TAs and peers, we use the digital signature as a user authentication method without protecting the TAs privacy because we do not need to hide the identity of our protocol authorities. However, when it comes to interactions between voters and peers, we need to preserve voters' privacy by blinding their signatures. The privacy-preserving consensus adds two steps:

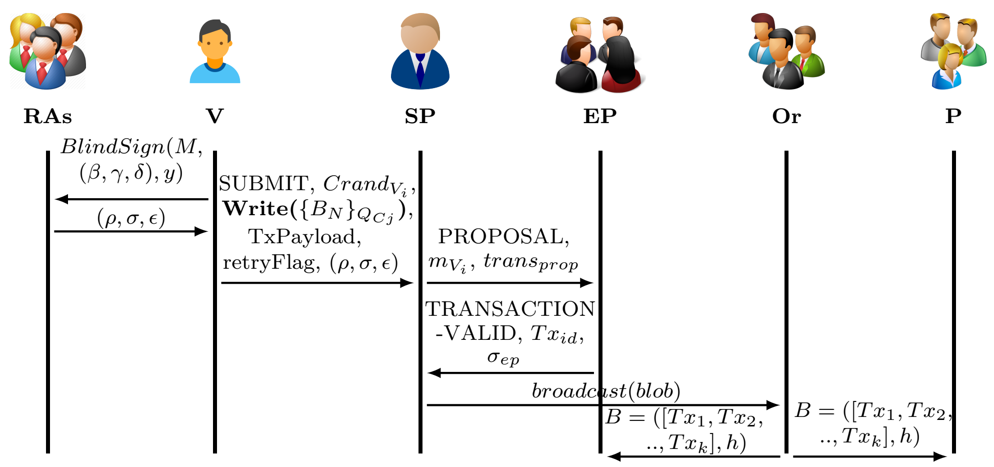

RAs
V
SP
EP
Or
P
$BlindSign(M, (\beta,\gamma,\delta), y)$
(ρ,σ,ϵ)
SUBMIT, $Crand_{V_i}$, **Write**($\{B_N\}_{Q_{Cj}}$), TxPayload, retryFlag, (ρ,σ,ϵ)
PROPOSAL, m_{V_i}, $trans_{prop}$
TRANSACTION -VALID, Tx_{id}, σ_{ep}
$broadcast(blob)$
$B = ([Tx_1, Tx_2, .., Tx_k], h)$
$B = ([Tx_1, Tx_2, .., Tx_k], h)$

Figure 4.8 – Interactions between eligible voters and BlindCons peers.

i) The signature of each eligible voter is blinded automatically after the vote is cast by the function $BlindSign(M, (\beta,\gamma,\delta), PK_{RA})$, where $M = (Credential_{V_i}|| Write(Enc\_Vote)||Enc\_Vote||retryFlag)$ is the message to be signed, (β,γ,δ) are secret values randomly chosen by the voter and PK_{RA} is the public key of the RAs.

ii) RAs blind the signature of each eligible voter by providing him with the tuple (R, S), allowing the voter to construct his/her blinded signature (ρ,σ,ϵ) to be used during his/her interactions with the peers.

The other steps are the same as the BFT based consensus, but instead of sending their signatures, the voters send their blinded signatures provided by the RAs.

1. **Initiating transaction:** <SUBMIT, $Credential_{V_i}$, $Write(Enc\_Vote)$, Enc_vote, retryFlag, (ρ,σ,ϵ) >
2. **Transaction Proposal:** <PROPOSAL,$m_{V_i}, trans_{prop}$ >
3. **Transaction Endorsement:** < TRANSACTION-VALID,Tx_{id}, σ_{ep} >
4. **Broadcasting to consensus:** broadcast(blob)
5. **Commitment:** $B = ([Tx_1, Tx_2, \ldots, Tx_k], h)$

The voters who intend to verify that their votes were properly counted must memorise the counter-values corresponding to their chosen candidates.

4.3.3.4 Tallying Phase:

After all votes have been cast; TAs proceed to the tally. We have as many TAs as candidates. Each tallying authority TA_k is responsible for counting the number of votes for a specific pseudo ID C_j: for example, the first tallying authority TA_1 decrypts, with its secret key $S_1 \cdot Q_{C_1}$, all bulletins that were encrypted with the public key Q_{C_1} (certainly these ballots contain votes for candidates with $C_j = 0$). TA_k starts by initiating a transaction to read encrypted votes from the Blockchain. This transaction passes through the five steps of the BFT based consensus. Then, it decrypts the votes with its secret key S_k that were encrypted with Q_{C_j} to reveal the bulletin number BN. Then, it reconstructs the ballot, identifies the chosen candidate, and added to the corresponding counter. At the end of this phase, TA_k publishes the count for each candidate using the following formula: $\sigma_{k,name_j} = \sum_{i=1}^{l_j} S_k \cdot Q_{BN_i}$; Where l_j is the number of votes received by the candidate j, S_k is the private key of the tallying authority k, $Q_{BN_i} = H_1(BN_i)$ and BN_i is the ballot number of the vote i that corresponds to the candidate with name $name_j$.

4.3.3.5 Verification Phase:

This phase allows voters to check that their votes were counted as cast and that the election final result corresponds to the sum of all eligible votes. It includes two sub-phases. During the first one, TAs calculate the list of chosen counter-values from each ballot number and the chosen candidate's name and publish this list on the Blockchain. Each eligible voter can read this list and verify the existence of his/her counter-value to be sure that his/her vote was counted correctly. The second sub-phase uses the homomorphism of pairings to check the accuracy of the tally. Using the published counts and the reconstructed counter-values, we can verify that the announced result corresponds to the sum of all eligible votes, as follows :

$$\begin{aligned}\prod_{i=1}^{l} CV_{BN_i} &= \prod_{k=1}^{m}\prod_{j=1}^{m}\prod_{i=1}^{l_j} CV_{BN_{i,name_j},k} = \prod_{k=1}^{m}\prod_{j=1}^{m}\prod_{i=1}^{l_j} e(Q_{name_j}, S_k \cdot Q_{BN_i}) \\ &= \prod_{k=1}^{m}\prod_{j=1}^{m} e(Q_{name_j}, \sum_{i=1}^{l_j} S_k \cdot Q_{BN_i}) = \prod_{k=1}^{m}\prod_{j=1}^{m} e(Q_{name_j}, \sigma_{k,name_j})\end{aligned} \tag{4.1}$$

Where $l = \sum_{j=1}^{m} l_j$ is the total number of votes. These equalities use the bilinear property of pairing: $\prod_{i=1}^{l_j} e(Q_{name_j}, S_k \cdot Q_{BN_i}) = e(Q_{name_j}, \sum_{i=1}^{l_j} S_k \cdot Q_{BN_i})$

4.4 Security Evaluation of DABSTER

Thanks to the use of the BFT based consensus, the BlindCons and a variety of cryptographic primitives, our protocol ensures several security properties. We discuss the security properties ensured by our protocol and prove, formally, that our solution guarantees vote secrecy, vote privacy, and voter's authentication.

4.4.1 Informal Security Evaluation

We evaluate our protocol according to a list of security properties that must respect a secure and practical voting system.

- **Eligible voter:** The registration and the validation phases of our protocol ensure that only eligible voters participate in the voting process. During the registration phase, RAs verify the identity of each voter via a face to face meeting and only eligible voters are provided with credentials. During the validation phase, RAs send the list of registered voters to the consensus peers, which are composed of voters, election organisers and observers, to verify the eligibility of all registered voters and validate or reject this list.

- **Individual verifiability:** This property is ensured by our protocol because our ballot structure includes counter-values. These values serve as receipts for voters and enable them to verify that their votes have been cast as intended without disclosing who they voted for. In fact, counter-values are calculated using the following formula: $CV_{BN,name_j,k} = e(Q_{namej}, S_k \cdot Q_{BN})$. Thus, we cannot get the candidate's name from the value of $CV_{BN,name_j,k}$.

- **Universal verifiability:** From the parameters published by the TAs during the verification phase, everyone can verify the accuracy of the final result by checking the equation (4.1).

- **Vote-Privacy:** This property is ensured thanks to the BlindCons. Before interacting with the consensus peers, RAs blind the signature of all eligible voters to hide their real identities. Voters' transactions are signed by the blind signature issued by the RAs and not with the voter's signature. Thus voters' identities are kept private and no one can link a vote to a voter.

- **Receipt-freeness:** In our case, a voter cannot find his/her vote from the counter-value $CV_{BN,name_j,k}$ and the other public parameters. He/she cannot therefore prove that he/she voted for a given candidate.

- **Coercion resistance:** Our protocol is not resistant to coercion. A coercer can force a voter to vote for a certain candidate and check his/her submission later using the counter-value.
- **Integrity:** The BFT based consensus and the blind signature algorithm prevent votes from being altered while keeping the voter's secrecy. Each transaction is stored in the Blockchain after being validated by $50\% + 1$ of the endorsing peers. This eliminates the risk of modifying transactions before storing them. We mention here that the BFT consensus is based on the assumption that 2/3 of the endorsing peers are honest.
- **Fairness:** During the voting phase, each eligible voter encrypts his/her ballot number BN with $Q_{C_j} = H_1(C_j)$ where C_j is the pseudo-ID of the desired candidate. Ballot numbers are secret and candidates' pseudo-IDs do not reflect the real identities of candidates thanks to the offset value, so that nobody can identify the chosen candidate from the encrypted vote. Thus, we cannot get partial results before the official count.
- **Robustness:** Our scheme is resistant to the misbehavior of dishonest voters who cannot invalidate the election by casting an invalid vote or by refusing to cast a vote.
- **Vote-and-go:** Our protocol does not need the voter to trigger the tallying phase; they can cast their votes and quit before the voting ends.
- **Voting policy:** DABSTERS allows the possibility to eligible voters to vote more than once and only their last votes are counted. It means that we have a maximum of one vote per voter in the final tally. Every eligible voter has a unique valid credential which is sent with his/her vote in the transaction.

4.4.2 Formal Security Evaluation

ProVerif is a fully automated and efficient tool to verify security protocols. It is capable of proving reachability properties, correspondence assertions, and observational equivalence. To perform an automated security analysis using this verification tool, we model our protocol in the Applied Pi-Calculus [ABF18] which is a language for modelling and analysing security protocols. It is a variant of the Pi-Calculus extended with equational theory over terms and functions and provides an intuitive syntax for studying concurrency and process interaction. The Applied Pi-Calculus allows us to describe several security goals and to determine whether the protocol meets these goals or not. To describe our protocol with the

Applied Pi calculus, we need to define a set of names, a set of variables and a signature that consists of the function symbols used to define terms. These function symbols have arities and types. To represent the encryption, decryption, signature, blind signature and hash operations, we use the following function symbols: `pk(skey)`, `aenc(x,pk(skey))`, `adec(x,skey)`, `spk(sskey)`, `sign(x,sskey)`, `checksign(x,spk(sskey))`, `BlindSign(x,smkey)`, `checkBlindSign(x, spk(smkey))`, `H1(x)`. Intuitively, the `pk` function generates the corresponding public key of a given secret key, `aenc` and `adec` stand, respectively; for asymmetric encryption and asymmetric decryption, `aenc` and `adec` follow this equation: `adec(aenc(x,y),pk(y))=x`. The `spk` function generates the corresponding public key of a given signature secret key, `sign` and `checksign` provide, respectively, the signature of a given message and the verification of the signature. They respect the following equation: `checksign(sign(x,y),spk(y))=x`. `BlindSign` and `checkBlind- Sign` stand, respectively, for blind sign and check blinded signature, `BlindSign` and `checkBlindSign` follow the equation `checkBlindSign(BlindSign(x,y),spk(y))=x`. We also assume the hash operation which is denoted with the function `H1`.

Because of the limitation on the number of pages, we put all ProVerif codes online[2] and give only the queries, the results of executing these codes, and the time it takes ProVerif to prove the properties in Table 4.3 (Execution times are expressed in seconds).

Property to evaluate	**Description**	**Result**	**Exec time**
Vote secrecy	To capture the value of a given vote, an attacker has to intercept the values of two parameters: the ballot number BN and the pseudo ID of the chosen candidate Cj.	Proved	0.012s
Voter's Authentication	We use correspondence assertion to prove this property.	Proved	0.010s
Vote privacy	To express vote privacy we prove the observational equivalence between two instances of our process that differ only in the choice of candidates.	Proved	0.024s

Table 4.3 – ProVerif results and execution times.

4.4.3 Blockchain Security Evaluation

DABSTERS Blockchain protocol has the following security properties.

4.4.3.1 Consistency:

A Blockchain protocol achieves consistency if can ensure that each valid transaction sent to the network will stay immutable in the Blockchain.

[2] `http://sancy.univ-bpclermont.fr/~lafourcade/DABSTERS_FormalVerif/`

Definition 4.1 (Consistency)**.** *A Blockchain protocol* $\mathbb{P}$ *is* $T-consistent$ *if a valid transaction* tx *is confirmed and stays immutable in the Blockchain after* $T-round$ *of new blocks.*

Theorem 4.1. *DABSTERS Blockchain protocol is 1-consistent.*

Proof. The consistency is achieved by agreeing on the validity of the transaction through a Byzantine Agreement process. Hence, the probability of not settling it in a new block is negligible if the transaction has at least $50\% + 1$ of a valid endorsement and the network has at most $\lfloor \frac{an-1}{3} \rfloor$ out of total n malicious peers, as it has been shown in [CL99] [LM07] under the terminology of safeness. The protocol achieves consistency after a new block is created (1-consistency) due to the chain is growing without forks. □

4.4.3.2 Liveness:

A consensus protocol ensures liveness if an honest client submits a valid transaction and a new block is generated with the transaction in it. Hence, the protocol must ensure that the Blockchain grows if valid clients generate valid transactions.

Definition 4.2 (Liveness)**.** *A consensus protocol* $\mathbb{P}$ *ensures* liveness for a Blockchain C *if* $\mathbb{P}$ *ensures that after a period of time t, the new version of the Blockchain* C' *is* $C' > C$*, if a valid client* $c_{i_{bc}}$ *has broadcasted a valid transaction* tx_i *during the time t.*

Theorem 4.2. *DABSTERS Blockchain protocol achieves liveness.*

Proof. Our protocol is a BFT-based consensus. Thus, liveness is achieved if after the transaction validation process, the network agrees in new block B with the transactions broadcasted by the clients during a period of time t. Hence, for valid transactions tx_i, where $i \in \mathbb{N}_0$, issued by valid a client c_i during a period of time t, the probability that $C' = C$ is neglected if we have at most $\lfloor \frac{n-1}{3} \rfloor$ out of total n malicious peers [CL99]. □

4.4.3.3 Blindness:

We use the definition of *blindness* defined by Schnorr in [Sch01]. A signature is properly blinded if the signer cannot get any information about the signature if the receiver follows the protocol correctly.

Definition 4.3 (Blind signature)**.** *A signing scheme is* blind *if the signature* $(m, \rho, \sigma, \epsilon)$ *generated by following the protocol correctly, is statistically independent of the interaction* (a, e, R, S) *with that provides the view to the signer.*

Theorem 4.3. *Okamoto-Schnorr signature* $(m, \rho, \delta, \epsilon)$ *is statistically independent of the interaction* (a, e, R, S) *between the authority* A *and the user.*

Proof. We recall how the protocol works. To generate a blind signature $(m, \rho, \sigma, \epsilon)$ the user chooses randomly $(\beta, \gamma, \delta) \in \mathbb{Z}_q$ to respond to the commitment a generated by A with the challenge $e = H(m, ag^{\beta}h^{\gamma}y^{\delta}) - \delta \bmod q$. The authority A then sends $(R, S) = (t - er, u - es)$ to finally obtain the signature by calculating $(\rho, \sigma) = (R - \beta, S - \gamma)$. Hence, for the constant interaction (a, e, R, S) and a unique set (β, γ, δ) randomly chosen per signature, we generate a signature $(m, \rho, \delta, \epsilon) = (m, R - \beta, S - \gamma, e + \gamma)$ that is uniformly distributed over all the signatures of the message m due to the random $(\beta, \gamma, \delta) \leftarrow \mathbb{Z}_q$ [Sch01]. $\square$

4.5 Conclusion

We proposed a fully decentralised electronic voting system that combines several security properties. This protocol, called DABSTERS, uses a new architecture that enhances the security of e-voting systems and guarantees the trustworthiness required by voters and election organisers. DABSTERS is designed to be implemented on private Blockchains and uses a new blinded signature consensus algorithm to guarantee vote integrity and voter's privacy due to the blinded signature's unlinkability property.

Chapter 5

Generating Functionally Equivalent Programs Having Non-Isomorphic Control-Flow Graphs

Contents

One of the big challenges in program obfuscation consists in modifying not only the program's straight-line code (SLC) but also the program's *control flow graph* (CFG). Indeed, if only SLC is modified, the program's

CFG can be extracted and analyzed. Usually, the CFG leaks a considerable amount of information on the program's structure.

In this work we propose a method allowing to rewrite a code P into a functionally equivalent code P' such that CFG(P) and CFG(P') are radically different.

5.1 Introduction

In the white-box security model, adversaries have access to a program's internals - assembly code, memory, etc. This model captures real-world attacks against low-end devices, as well as software disassembly and dynamic analysis. Such attacks may allow the adversary to extract secrets from the implementation, either in the form of tokens (passwords, etc.), intellectual property (algorithms, etc.), or may help uncover design flaws that may later be exploited. Reverse-engineering may also help the adversary recognize some traits that the program shares with other programs, e.g. in the case of malware analysis or intellectual property infringement. The general aim of obfuscation is to prevent reverse-engineering, by defeating automated methods and stave off human efforts to make sense of the code. Applications of RE-evasion techniques are many, and constitute for instance an essential building block of digital rights management (DRM) systems.

Historically, program identification focused on finding known code chunks called *signatures* in the binary. While this technique is still widely in used amongst intrusion and virus detection systems, such an approach requires both extensive, and up-to-date, databases (to account for the ever-growing corpus of threats) and a very efficient binary comparison method. At the same time, widely used packagers with self-modifying code capacity, now standard amongst virus designers, made the traditional signature-based approach less and less effective.

Indeed, an increasing number of malicious programs rewrite their executable code not to feature any recognizable code of significant length. In practice, it is unnecessary to resort to very complex rewriting mechanisms: the malicious code can simply add (or remove) useless instructions or instruction sequences (such as `nop`, and reversible register operations, e.g. `inc`/`dec`) to thwart a trivial comparison. While such variations can be accounted for, they require significantly more effort from the analyst, especially when scanning a large number of files.

An alternative, and certainly complementary approach to malware detection and analysis consists of running the program under a controlled environment, or *sandbox*, Thus, every operation can be monitored and does not impact the "real" underlying system. Sandboxes typically implement a form of a virtualised environment, and monitor access to resources, secrets, and peripherals to detect abnormal behaviour. Naturally, the term "abnormal" is application-dependent; hence, this approach assumes that characteristic behavioural features are known and are sufficiently distinguishable from those of uninfected software. Moreover, managing such a controlled environment is demanding in terms of resources and time, limiting the interest in sandboxing as a program identification tool.

Recent research focused on methods for comparing programs using control-flow graph isomorphism [DR05, Fla04, KKM$^+$05]. The rationale is that the program's flow graph (CFG) wouldn't be altered significantly by the adjunction or removal of useless "decoy" operations, the kind of which thwarts direct comparison. CFG comparison techniques are also unaffected by straight-line code obfuscation techniques, e.g. when each function's code is completely rewritten. CFGs can be extracted statically to a large extent, and therefore constitute an attractive and resource-frugal alternative to full-blown virtualisation.

5.1.0.0.1 Defeating CFG analysis. In the malware-writing community, a typical anti-reverse engineering technique is the *trampoline*. Instead of using typical control flow instructions such as `jmp` or `call`, the program makes heavy use of exception handling, that preempts the instruction pointer and runs the exception handler, which redistributes control flow to another program part (see e.g. [Dav15]). After execution, each program part raises an exception, and falls back to an exception handler (hence the name, trampoline). There can be several trampolines, which may be created and moved at runtime, and code boundaries need not be rigid. This prevents disassemblers from reliably cross-referencing information and makes it difficult to perform dynamic analysis as well, because it is typically impossible to run such code within a debugger.

However, because there is no classical call hierarchy, trampolines must emulate the stack, and an analyst that recognises the mechanism can easily reconstruct the control flow graph by following this pseudo-stack. Therefore, while trampolines slows down analysis, it is by no means an efficient method anymore against trained reverse engineers, and the additional effort put into designing such code is not worth the marginal gain.

Recent work tried to automate the process, which strives to achieve a "flat" control flow graph, i.e. a graph with either a single central trampoline that dispatches execution, or a program that is fully unrolled and appears as a long straight-line code segment without internal structure [WHKD00, CGJZ01, LD03, PDA07, LK09, CP10, SK11]. However not only are such techniques not always applicable, but more importantly they tend to produce code that has salient signatures while "flat".

5.1.0.0.2 Our contribution. This work addresses the question of rewriting a program in a way that hides its original control flow graph from static analysis (and, to a certain extent, from dynamic analysis as well), while preserving functionality. Straight-line code (SLC) obfuscation techniques can be used to destroy remaining signatures on top of our construction. Indeed, SLC obfuscators have already been described in the literature and shown to defeat classic code analysis techniques effectively[SKK$^+$16]. The rewriting is randomized, and produces different outputs every time. Unlike the trampoline construction, whose heavy use of exception handling is easily recognizable, and from there, traceable, our construction only uses common instructions and relies on a specific routing mechanism along execution — which is much harder to detect.

More formally, given a program P, we show how to obtain a functionally equivalent program P', such that the CFG of P' is essentially a random graph. This transformation is automatic, and we show how to implement a CFG-transcompiler for the x86-64 architecture, which is widely used and furthermore makes our implementation easier.

5.2 Control Flow Graph Transcompilation

5.2.1 Prerequisites

The control flow graph of a program is a graphical representation, based on nodes and edges, of the paths that the program might traverse during its execution.

Definition 5.1 (Control Flow Graph)**.** *The (full)* control flow graph *of a program P is the graph whose nodes are the program's instructions and the edges are control flow transitions. The* restricted control flow graph *of P has for nodes* straight-line blocks*, i.e. a maximal sequence of code without departure or arrival of static jumps, and there is an edge from node x to node y (and we write $x \to y$) if either of the following conditions holds:*

- *The code of node y is located immediately after the node x, and a conditional jump separates both.*
- *The last instruction of the node x is either a conditional or a static jump, which is a call to the physical address of the beginning of the node y.*

In the following, unless specified otherwise, we always refer to the *restricted* control flow graph. This construction does not include information about dynamic jumps: In practice it is challenging to statically and reliably resolve dynamic jumps. Therefore, the `ret` instruction, which we cannot ignore since it is often used to implement function calls, will be dealt with in a special way.

However, other dynamic and indirect control flow modifications (e.g. by direct alteration of the instruction pointer, or non-standard exception handling) are not considered in this work. On the one hand, this is limitation may prevent some programs from undergoing the transformation that we propose. On the other hand, this may constitute an interesting countermeasure against code-reuse and hijack attacks that leverage such possibilities.

Let P be the program to be obfuscated. Then, we denote by $G = (V, E)$ the CFG of P, where V and E correspond respectively to the nodes and edges of G. Let $G' = (V', E')$ be a given "final" target CFG.

Example 5.1. *Consider the following program, implementing a simple double-and-add algorithm:*

```
dbl_add(int, int):                    ; Compute ab from integer arguments a and b
        test    esi, esi
        mov     eax, 0                ; tmp = 0
        jle     .end                  ; if b == 0, return tmp
.loop:
        lea     edx, [rax+rax]        ; tmp2 = 2 tmp
        add     eax, edi              ; tmp = tmp + a
        test    sil, 1
        cmovne  eax, edx              ; if b even set tmp = tmp2
        sar     esi                   ; shift b to the right
        jne     .loop                 ; loop if b > 0
        rep ret
.end:
        rep ret
```

The CFG associated with this program is represented in 5.1, where the instructions' arguments have been removed for clarity. The associated restricted CFG is represented in 5.2.

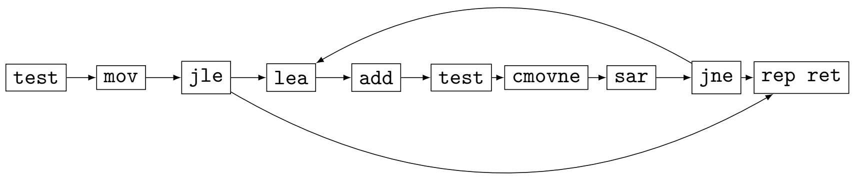


Figure 5.1 – Full CFG of the program of 5.1.

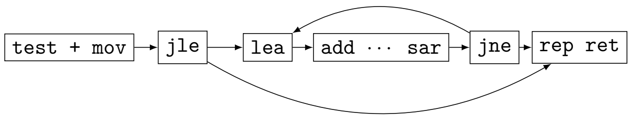


Figure 5.2 – Restricted CFG of the program of 5.1.

5.2.2 Overview of our Approach

Our goal is to rewrite P into a program P' that achieves the same functionality as P, but whose CFG is $G' \not\simeq G = (V, E) = \mathrm{CFG}(P)$. This is achieved in successive steps, illustrated in 5.3, 5.4, 5.5 and 5.6.

5.2.2.0.1 Step 1: Relabeling. We start from a morphism π between the two graphs, i.e. a function that is injective on nodes and preserves edges. If we fail to find enough nodes or edges to perform this operation, which happens with a very low probability when the target graph is large enough, we simply start over with a new random graph G'. The process is illustrated in 5.3.

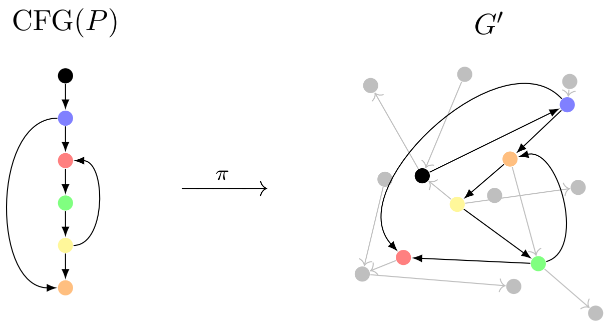


Figure 5.3 – Illustration of Step 1: Relabeling. The original nodes and edges from CFG(P) are assigned different colors; other nodes are in gray.

5.2.2.0.2 Step 2: Breaking Edges. Then, additional nodes will be added by transforming the graph. The idea is to replace simple edges by paths in $G' = (V', E')$, i.e. for each edge $(a, b) \in E$, corresponding to an edge $(\pi(a), \pi(b)) \in E'$, we replace $(\pi(a), \pi(b))$ by a path $(\pi(a), f((a, b)), \pi(b))$, where f is a prescribed function. Such a function $f : E \to \mathrm{List}(V')$ must return paths already present in G', i.e. assuming that $f((a, b)) = (s_1, \ldots, s_n)$,

- $(\pi(a), s_1) \in E'$
- $(s_n, \pi(b)) \in E'$
- $\forall i \in \{1, \ldots, n-1\}, (s_i, s_{i+1}) \in E'$

We keep track of which edges were originally present and which edges were added at this step. The process is illustrated in 5.4.

5.2.2.0.3 Step 3: Identify Active and Passive Nodes. The previous step introduced "extra" operations between a and b. Since we wish to preserve the original program's functionality, we should make sure that

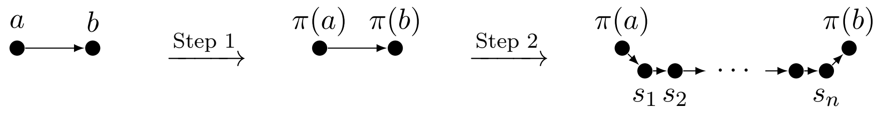


Figure 5.4 – Illustration of Step 2: Breaking edges. The original path $\pi(a) \to \pi(b)$ is extended by a path $f((a, b)) = (s_1, \ldots, s_n)$ of G'.

only the original endpoints, a and b, are executed, while all the intermediary nodes are without effect when executed. Thus, we call a and b the *active* nodes, and the intermediary nodes (i.e. nodes that do not exist in the original CFG) are called *passive*.

Remark 5.1. *A node that is neither active nor passive in the control flow graph G can be considered either active or passive in G'.*

Depending on the execution path taken, some nodes may be active or passive (e.g. 5.5). To decide whether a given node is active or passive, the program (more precisely, the node itself) checks at runtime the value of a routing variable (see below).

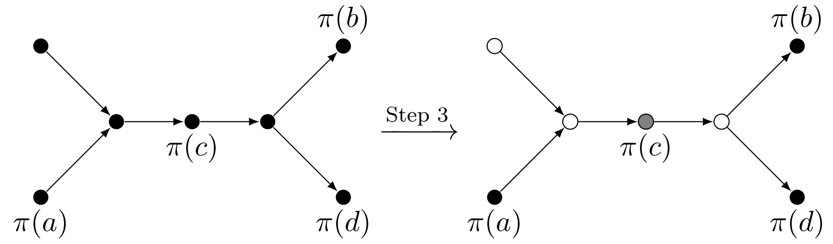


Figure 5.5 – Illustration of Step 3: Identifying active and passive nodes. Here two original sequences $\pi(a) \to \pi(b)$ and $\pi(c) \to \pi(d)$ cause some nodes to be passive (empty circle), active (filled black circle), or active depending on the execution path taken (grey circle).

5.2.2.0.4 Step 4: Routing. Finally, we transform each node so that the execution of passive nodes is without side effects (a process we call *passivation*), except continuing through the sequence of nodes until an active node is attained. To that end we introduce an additional "routing" variable that will be updated as the program is executed (e.g. 5.6).

Nodes consult the routing variable to know whether they are active or not; if not, they simply hand over execution to the next node in sequence (possibly after executing dummy instructions).

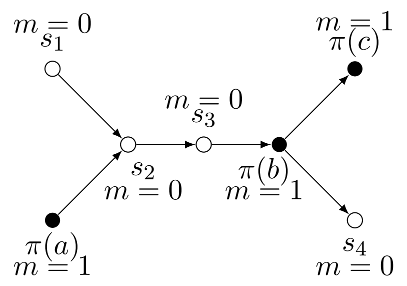


Figure 5.6 – Illustration of Step 4: The path is taken according to the routing variable m. If the node is passive ($m = 0$), the path to be taken will be the subsequent node. In the case of an active node ($m = 1$), the next node will be defined by the current node

5.2.3 Contexts

During program execution, every node in the transformed program undergoes the following procedure:

1. Determine whether the node is active or passive.
2. If active, restore the registers. Otherwise passivate itself.
3. Run the code.
4. Call the next node in the sequence.

To allow this series of operations, we introduce the concept of *contexts*.

A context is a set of variables that save the node's state, in a way that can later be restored. Thus, each traversed node is associated with a context available just when the node is traversed.

Since passive nodes do not suffer side effects, they cannot in particular find the next node to be called; hence the next node is part of the context. On the other hand, if the node is active, it may ignore this part of the context and branch itself to another destination.

5.2.4 Node Passivation

Node passivation requires us to cancel the instruction(s) being executed, or compensate for their effects somehow. We do this by using both the registers and the stack (it is not possible to rewrite registers that are in active mode), leveraging the specificities of the x86-64 architecture.

5.2.4.0.1 Register operations. Any register operation can be dealt with by using contexts, except for the stack registers.

5.2.4.0.2 Stack operations. Stack operations are harder to compensate: the following instructions affect the stack

`PUSH, POP, PUSHA, POPA, PUSHAD, POPAD, PUSHF, POPF, PUSHFD, POPFD`

We control writing and reading in the stack by using a pointer to a "trash" address, stored as a fixed value. If a passive node attempts to write something in the stack, we redirect the address to the trash, nullifying the instruction's effects. The reading process is handled in the same way. If the node is active, the real address is used.

The context m is used in the following way: after a `PUSH`, we perform the following operation to the pointer to the top of the stack p:

$$p \leftarrow p + (8 \ \& \ m)$$

where

- $m = 1 \cdots 1_2$ if the node is passive. In this case the operation will be compensated and will not have any effect due to the top of the stack not changing.
- $m = 0$ if the node is active. In this case the addition is useless and the `PUSH` works as intended.

5.2.4.0.3 `MOV` instruction. `mov` instructions from one register to another are already without effect, since register values are restored at the beginning of each active node, and are stored in the environment. However, `mov` instructions that involve a memory address require additional care, and we use the same technique as for the stack: the address is rewritten to the "bin" when the node is passive. The transformation follows this:

$$\text{address} = (\text{address} \ \& \ (\neg m)) | (\text{trash\_address} \ \& \ m)$$

This technique also hides the addresses that are really used during program execution.

5.2.4.0.4 Function calls. We will distinguish *library* function calls and calls to *internal* functions, that are defined in the code.

5.2.4.0.5 Library calls. In the case of library function calls, each of them is treated separately by using a specific context per function. Now, considering that it is impossible to handle all the functions simultaneously, we propose to call the functions by using parameters that make them ineffective.

Example 5.2. *In the following "Hello World" program, where we make ineffective the function printf by loading to EAX the address of an empty sentence (auxiliary parameter) and set the stack pointer to the address of EAX.*

```
extern _printf
global _main

section .data
param1: db "Hello World",10,0
paramaux: db "",0                  ; declaration of the empty sentence

section .text
_main:
    push param1
    lea eax, [paramaux]            ; paramaux address placed in EAX
    mov [esp], eax                 ; pointer to the empty sentence
    call _printf
    add esp,4
    ret
```

5.2.5 Jumps and Internal Calls

5.2.5.0.1 Internal calls. Recall that we distinguish between a call to an address in a `PUSH` from a static jump. This makes the above transformation effective to handle these instructions. However, the `RET` instruction corresponds to a dynamic jump and is subtler to handle.

Let n be a node with a `RET` instruction in G, and assume that in G' the corresponding node $\pi(n)$ has two neighbors, f_1 and f_2. Their addresses are fixed, so that one can place, on the top of the stack, the address of the node that follows $\pi(n)$ (either f_1 or f_2).

Example 5.3. *In the following example, we print on the screen the result returned by* **func1**. *In this case, we jump from* **func1** *to* **func2**, *adding the desired address on the top of the stack by using a* **push** *operation. As a result, the program jumps to* **func2** *instead of jumping back to the address after the call.*

```
extern _printf
global _main

section .data
num DD 2,3
format: dd "num: %d" , 10, 0

section .text

_main:
  mov eax,0             ; eax = 0
  mov esi, [num]        ; edi = 2
  mov edi, [num+4]      ; esi = 3
  push esi              ; pass param 3 to .func1
  push edi              ; pass param 2 to .func1
  push eax              ; pass param 1 to .func1
  call .func1           ; jump to func1
  add esp,12              ; pop edi, esi and eax from the stack

  push eax
  push dword format
    call _printf          ; print eax in the screen
  add esp,8               ; pop stack 2*4-byte

.func1:
  push ebp
  mov ebp,esp             ; set stack base pointer
  sub esp, 4              ; creates space for one 4-byte local variable
  push edi                ; Save the values of the register that the function will use
  push esi
  mov eax,[ebp+8]         ; move param 1 to EAX
  mov edi,[ebp+12]        ; move param 2 to EDI
  mov esi,[ebp+16]        ; move param 3 to ESI

  mov [ebp-4],edi         ; var local = 2
  add [ebp-4],esi         ; var local = 5
  mov eax, [ebp-4]        ; EAX = 5

  pop esi                 ; remove esi from the stack
  pop edi                 ; remove edi from the stack
  mov esp,ebp
  pop ebp                 ; takedown stack base pointer
  lea ecx,[.func2]
  push ecx                ; push func2 address on the top of the stack
  ret                     ; jump func2

.func2:
  push ebp
  mov ebp,esp             ; set stack base pointer
  sub esp, 4              ; creates space for one 4-byte local variable
  push edi                ; Save the values of the register that the function will use
  mov edi,[num]           ; edi = 2

  mov [ebp-4],eax         ; var local = 5
  add [ebp-4], edi        ; var local = 7
  mov eax,[ebp-4]         ; EAX = 7

  pop edi                 ; remove edi from the stack
  mov esp,ebp
  pop ebp                 ; takedown stack base pointer
  ret
```

5.2.6 Routing

Once we have passed through a passive node, without changing the environment, we must be capable of taking the next desired branch. As each node is a maximum of two out-degree, all that we need is a boolean variable in the environment that will indicate to which child we must to go.

In practice, it is enough to maintain a global routing variable r. This allows the sequence of branches to follow (left or right) between two consecutive nodes. Hence, we modify r for each active node found and its i-th bit gives the direction of the i-th branch of the current path. We will denote by r_i the i-th bit of r.

Remark 5.2. *Routing variables have a limited size if we use native types, it is straightforward to extend them but additional arithmetic is needed.*

5.2.6.0.1 `JUMP` instruction. First, we need to transform a conditional `jmp` from P into a `jmp` that goes to the next node as determined by r_i. For simplicity, we assume that all conditional jumps test a “zero flag”, which is set by a comparison just before the jump. For example, we have the node A (with children B and C) and the following program:

```
cmp (...)      ; comparison
je B           ; conditional jump to B
C              ; next node
```

We can save the routes in some constants A_to_B and A_to_C as we know how to move from node A to node B or C in advance. For doing so, we use the following code:

```
mov routing_variable, A_to_C    ; set routing variable
cmp (...)                       ; comparison
cmove routing_variable, A_to_B ; set routing variable iif comparison succeeds
```

This program then jumps according to the first value of the routing variable.

Note that, for passive nodes, routing variables are set to the (masked) trash address.

5.2.6.0.2 `RET` instruction. When a node is passive, we want to have two possible branches as in the case of the jump instruction. To achieve this, we also store the constants A_to_B and A_to_C, and we will use the mask m as the context. We will go to node B if $r_i = 1$ and to node C if $r_i = 0$.

We want to put at the top of the stack the address to which we want to go. Hence, we just add the following line before the `ret`:

$$p \leftarrow (p \ \& \ m) | ((r \ \& \ A\_to\_B) | (\neg r \ \& \ A\_to\_C)) \ \& \ \neg m$$

The transformation presented above allows us to modify the program’s control flow graph. Now, we can transform an arch into a path, and ensure that the path’s execution is identical to the effect of running the arch in the original graph.

5.3 Control Flow Graph Obfuscation

While the presented construction effectively transforms the program’s CFG, the resulting construction has a strong signature, and it is easy to reverse the process to obtain the initial graph. It is indeed enough to run the program and identify nodes that change the routing variable. These nodes are the active ones, and it is possible to reconstruct the original control flow graph.

In this section we propose several ways to obfuscate the transformed program and make this reconstruction harder. First, we will "force" the program's execution to recover the initial control flow graph successfully; we then hide the nodes' activity, including the operations on the routing variable, which is a signature of an active node.

5.3.1 Forcing Execution

For now, we know that the routing variable suffices to determine the next active node. Therefore, we will modify its definition and use it to hide the control flow from static analysis. The routing variable is now maintained as a sequence of bits $(r_1, \ldots, r_n)$.

Upon transitioning to node i, we apply to the routing bit r_i a random permutation f_i of $\{0, 1\}$.

Example 5.4. *For example, if one seeks to obtain at the end of the function a bit equal to 1, the following operations can be used:*

```
$r \leftarrow 0$                      *Null routing variable*
$a \leftarrow \mathrm{rand}()$        *Introduce randomness*
$t \leftarrow 5a$
$t \leftarrow t + r \times a$
$r' \leftarrow t/a \bmod 2$
```

At the end of the code execution we obtain $r' = 1$. If we declare $r = 1$ instead, we get $r' = 0$.

We can easily generate the random flips f_i by using an arithmetic operation and its inverses. As determining the value of a variable is undecidable, running the program is the most natural way to get information about the execution paths taken.

5.3.2 Node Hiding

In the same way that routing bits are masked, we can hide the value of the bit indicating whether a node is active or passive. However, by doing so, the node i only hides the status of node $i + 1$. The mask's value can also be changed by choosing a random number between m and $\neg m$, and updating the formula accordingly.

5.3.3 Route Hiding

Updates of the routing variable are crucial, as they immediately reveal active nodes. To hide the information about the routing changes, we extend each path beyond the active node, and introduce a weak form of "onion" routing, where the next node is determined at runtime. The rationale is that determining whether a node is active or inactive will require recovering the full route leading to this particular node.

We introduce two additional variables per node, called *path* and *next path*. The *next path* variable is masked (XORed) with a value that depends on the node. Upon execution of an active node, the values of these two new variables are further modified.

If the next node is C, the route from B to C is stored in *next path*, and masked by being XORed with the constants of every intermediary node between A and B.

The number of hops is counted. Upon arriving at the final hop B of the path from A to B, we swap the *next path* and *path*.

The route hiding process is illustrated in Figure 5.7.

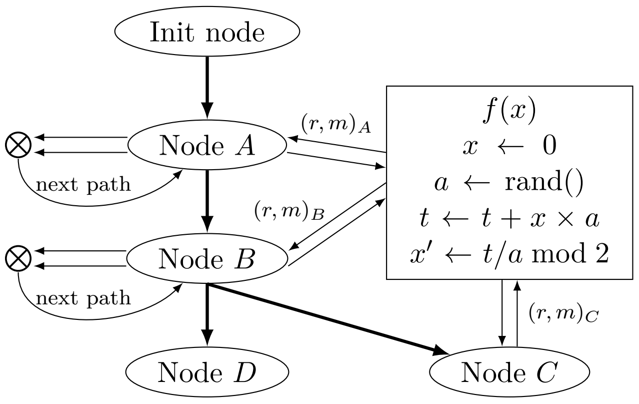


Figure 5.7 – Diagram of hiding process for nodes and routes

5.4 Security

Intuitively, the security of our construction depends on the hardness of identifying active nodes.

This can be formalized as an adversarial game, whereby a more precise security notion can be given:

CFG-FullRecovery Game:

1. The challenger provides a program CFG $G = (V, E)$
2. The adversary chooses a set $N \subseteq V$

The adversary wins the game if the nodes in N are the active ones.

To get a grasp on how hard this game is, assume that we choose N at random in V, where there are exactly $|N|$ active nodes:

$$\Pr\left[N \text{ is exactly the active nodes} \mid N \subseteq_R V\right] = \frac{1}{\binom{|V|}{|N|}} = \frac{|N|!(|V| - |N|)!}{|V|!}.$$

If one node out of two is active, and there are more than 42×2 nodes in V, this probability is negligible. Thus, we may hope, for realistically large programs, to resist adversaries for which there is no better way to choose N than selecting a random subset of V.

However, in practice, adversaries may succeed in recovering smaller portions of the CFG. This corresponds to the following game:

CFG-One Recovery Game:

1. The challenger provides a program CFG $G = (V, E)$
2. The adversary chooses a node $n \in V$

The adversary wins the game if n is active and n is not the first node of G (which is always active).

The success probability if n is chosen at random is

$$\Pr[n \text{ is active} \mid n \in_R N] = \frac{|N|}{|V|}$$

where again N is the set of (actually) active nodes. In the balanced case, where $2|N| = |V|$ this probability is exactly one half. When that is the case, and V is large enough, security in this second game implies security in the first game.

As discussed above, static analysis cannot in general determine the variables' values in a given node (by Rice's theorem [Ric53]). Given that the difference between active and passive nodes is only semantic; for a general program determining whether a given node is active is undecidable.

Hence, our obfuscation scheme should be secure against static analysis, for large enough values of N and few enough active nodes.

5.4.1 Security Against Dynamic Analysis

Dynamic analysis is performed by running and monitoring the program. As mentioned previously, the first node is always active. Therefore, the second node can be determined as follows: Execution continues until the *next path* variable is updated. Thus, we know that there is an active node between the current node B and the first node A.

The analyst then performs the following operation: For each node n between A and B in the CFG, replace n by another operation, and run the program up to B. Thus, there are at most $|V|$ nodes to test. In addition, a node is active if, when modified, the program's state at B has changed.

As each test is required to continue running the program until B, which can take up to $|V|$ steps, it is then possible to determine the next active node in $O(|V|^2)$. By running this procedure iteratively for all nodes, we reconstruct the list of active nodes, i.e. the original CFG, in $O(|V|^3)$ operations.

5.5 Implementation

Given as input a program CFG, we construct a "target" CFG to which the original program is mapped.

1. *Graph generation.* We generate a random graph with n edge and maximum out-dregee two, using a variant of the Tarjan-Eswaran algorithm [ET76, Rag05].

2. *Linearisation.* This graph is linearised, so that it corresponds to a CFG. For this purpose, we use the scheme presented by Leroy for the CompCert compiler [Ler15]. We then select a random morphism π between the initial graph and the new graph that we are creating.

3. *Transformation.* We begin the transformation by identifying the active and passive nodes. The edges for paths are then changed by neutralizing (passivation) the instructions using: registers and stacks operations, transforming the jumps and internal call, and defining the route to follow according to the routing variable. Finally, we remove the signature of the new graph hiding the routing variable and node's status (active or passive) by randomizing their values and adding the variables *path* and *next path.* To mask the variable *next path* we XOR it with the node's values.

The source code of this implementation is available from the authors upon request.

5.6 Conclusion

This work presents a control flow graph trans-compilation algorithm allowing to transform a program into a new functionally equivalent program. This algorithm uses common instructions such as register and stack operations, and a random routing variable, such that the resulting CFG is entirely different from the original one.

Part III

Tokenomics and Compliance in Blockchain

Chapter 6

A Novel Implementation of ERC20 Standards on Permissioned Ledger on a Practical Use Case

Contents

In this work, we present an ERC20 compatible token based on a permissioned ledger, called PlasticToken. PlasticToken is a currency promoting plastic reuse via the creation of a circular economy revolving around plastic reuse and revaluation. This currency has been implemented on top of the Hyperledger Fabric blockchain, fully supporting the ERC20 specifications.

After explaining the technical implementation, we expose how PlasticToken promotes local initiatives reducing plastic waste, and how coin generation is turned into a mechanism spreading awareness about plastic recycling. Another contribution of this work is the introduction of a cash-in and cash-out mechanism: it is possible to issue physical PlasticTokens, based on our virtual currency, and trade them, and convert them back to virtual assets, while maintaining the global security of the scheme.

PlasticToken is open-source, already operational, and is currently being tested by a few pilots in several European countries. The source code is open and accessible on Github (the link will be given after peer review).

6.1 Introduction

During the last years, the popularity of the blockchain and cryptocurrencies has been increasing. As a result, it has reached important notoriety in scientific and IT journals, and general public media. Moreover, the proliferation of the cryptocurrencies and the "cryptomania" experienced in the trading markets in 2017 has soared with the use of this technology for financial investing, reaching today a total market capitalisation of around $ 205 billion spread over more than 2.300 cryptocurrencies [Coi19].

The blockchain is considered a foundational emerging technology of the Fourth Industrial Revolution [HWW18] due to the capacity to enable assets transfer between users in a distributed manner and without a third trusted party. Therefore, blockchain technology democratises the financial market allowing the users to be part of it as an investor or as a service provider (e.g. miners). Nevertheless, blockchain can be implemented in several ways to provide solutions in different sectors like in healthcare [Con16a] [Med19], supply chain [IBM16], food quality control [Ibm19], among others. Although we can find several blockchain applications, there are just few use cases using blockchain to address environmental problems like carbon footprint or a new sustainable circular economy based on reuse and revalue recycled materials like plastic.

6.1.1 Plastic Reuse

Plastic use has skyrocketed over the last decades, and recently plastic waste has become a societal problem, weighting on the well-being, economy and environment. Hence, voices have emerged for a new plastic economic movement [FF17]. In addition, recent reports explore the possibility of a new economy in which used plastic is not a waste, but rather an asset [FFC16, Eco19]. New products are made out of single-use plastic, or used plastic components that would have otherwise been incinerated or buried, in this new economic model. Far from being wishful thinking, plastic-centered economics has gained wide acceptance by the public, as highlighted in Gartner's report [EARB16], citing numerous high-profile vendors committed to zero waste goals.

In parallel, national and international institutions are also starting to tackle the issue. This is, for instance, what the European Union is pushing forward, through its "plastic in the circular economy" strategy and roadmap [Com16]. On a much lower scale, many local initiatives have started to reevaluate plastic in the local and circular economy, thus reducing the amount of plastic waste, estimated to be around 25 million tons in the EU-28 [Eur].

Due to these preoccupation, our project PlasticTwist aims to incentivise citizens to reduce their plastic waste, by collecting plastic assets and recycling or reusing them, for instance through fablabs or other innovation centres. Furthermore, to foster the circular economy and keep the fidelity of its user, PlasticTwist relies on an electronic token, called PlasticToken, which this work focuses on.

6.1.2 Blockchain and Electronic Payments

The concept of electronic money can be traced back to 1983, when David Chaum's e-cash paper [Cha83] was published. Chaum's paper assumed a system in which a bank could digitally sign and allow transactions from any user, using blind signatures as to keep the user identity private. However, this scheme failed to gain traction as it required a constant access to the Internet, which was far from possible at that time. While not a critical caveat, the scheme still relies on a central authority - the bank.

In parallel, the first work on decentralised consensus –how different parties, with possible failures in their communications can reach an agreement– was published by Lamport in [LSP82]. In 2008, relying on these previous works as well as some others such as [BHS93b, HS91, DN93], the anonymous author(s) under the name of Satoshi Nakamoto invented the blockchain, a network in which all parties agree on financial transactions, without the need of a central authority which must be trusted by all parties [Nak08]. Based on this paper, Nakamoto implemented the first cryptocurrency, Bitcoin.

Simply speaking, a blockchain is a decentralised database in which new chunks of information (transactions) can be added, but none can be removed. Hence, one can reconstruct the current state of the blockchain by operating all the partial iterative transactions. Most of the time, new transactions are added by blocks, which are appended to the chain by miners: users who spend time-solving a cryptographic puzzle (for the proof of work), then publish the block so that other miners can acknowledge the result and mine on top of this block. For their work, miners are rewarded with newly minted coins, as well as the transaction fees, paid by the users asking the miners to add their transaction in the newly mined block.

Because it forms a peer-to-peer system in which any user can join, and mine, Bitcoin is the first decentralised currency: no central organisation is in charge of securing the transactions; the entire network is. As a matter of fact, an attacker willing to revert a transaction must control 51% of the computing resources of the network–which, in September 2019, is estimated to be around 10^{20} hashes per second [Dig]. However, this security comes at a price: because all miners compete to add a new block to the blockchain, much computational power is wasted to secure the blockchain. In June 2019, it was estimated that the Bitcoin network consumes at least 37 TWh per year, as much as the entire country of the Czech Republic [Dig]. This energy consumption is unavoidable, as Bitcoin users are anonymous and as such, not trustworthy: it is not possible, security-wise, to allow one user to append blocks to the chain easily. Other blockchains with a lower hash rate have been subject to a 51% attack over the last two years [Att19].

The energy consumption problem in Bitcoin is present in other public blockchain technologies. The main issue for this blockchain architecture is to achieve trust in an anonymous or pseudo-anonymous permissionless network. Hence, the decentralised database replication must have a proof of honesty. This can be achieved using proof of work (e.g. Bitcoin), where the miners spend CPU resources, that consume energy, to prove that they are honest. Or it can be used proof of stake, that consume less energy, where the miners need to spend their own asset to prove their honesty. Therefore, in a public blockchain architecture, the mechanism to achieve trust asks the network members (e.g. miners) to prove that they are honest generating inefficiencies, and making the transactional ecosystem more expensive.

This caveat has been solved by introducing consortium and private blockchains, such as Hyperledger Fabric [Fou19], developed by IBM. In these blockchains, only authenticated members can add transactions to the chain. This additional authentication lowers the required amount of work miners must provide; in fact, Hyperledger proposes several consensus algorithms that do not rely on the proof of work, but rather, for instance, proof of elapsed time or byzantine fault tolerance consensus.

The problem of energy consumption has also been tackled in some permissionless ledgers, who adopted other consensus modes, such as Proof of Space [ABFG14], proof of stake (see for instance [KRDO17, BG17]). However, while they solve the issue of energy consumption, these consensuses are not exempt from faults. For instance, proof of space replaces the race for energy for a race of memory capacity, and proof of stake, while elegant on paper, is difficult to implement securely. For instance, Ethereum has spent several years defining its proof of stake [BG17], which should soon be implemented.

6.1.3 Related Work

The first concept of using a blockchain for currency purposes is obviously Bitcoin [Nak08]. Furthermore, non-public ledgers have already been used in several cryptocurrencies for good behaviour incentives, such as with Eco-Coin [Eco18], where users can signal to an authority the eco-responsible actions they made. After validation, they get a financial reward in the form of crypto coins. However, this process is highly centralised, and every action, happening off-chain, must be validated by an authority. Similarly, the blockchain Quorum [Quo16] has implemented a permissioned blockchain relying on Ethereum. Still, as far as we know, no ERC20 token has been implemented on this blockchain, as it is usually more used for private businesses than public use and public awareness. Another blockchain project based on a permissioned ledger for cryptographic token implementation is the Social Plastic initiative [Ban19]. This project is promoted by the Plastic Bank and the Cognition Foundry. The Social Plastic initiative uses a native crypto token implemented on Hyperledger Fabric to be used as a rewarding mechanism for plastic recycling. However, this native token implementation restricts the capacity to be integrated the other token's economies, reducing the adoption rate due to the lack of flexibility and interoperability.

On the other side of the spectrum, alternative local currencies are getting more popular in the last decade. For instance, in France, more than 80 alternative currencies are listed in [Cit], with at least 20 of them launched in the last two years. Of course, these alternative currencies still remain marginal (with less than 1000 users each), but most of them advocate for more responsible use of one's money: as the currencies are local, they are only accepted by local shops, which in turn only buy from local producers. Hence, alternative local currencies promote local economies, and somewhat support a circular economy.

PlasticToken aims to offer a solution to both issues by merging them. This currency aims to promote a local and circular economy, with the support, security, and functionalities offered by consortium ledger blockchains.

6.1.4 Economic Environment

As this project aims to promote a circular economy, we need to design a place in which plastic can be exchanged and sold: currently, used plastic is mostly considered as a waste, and there is no platform aiming to reduce its disposal. The PlasticToken token is only a portion of the Plastic Twist project. The other parts focused, among other things, on creating a platform for hosting a marketplace for plastic.

Hence, the Plastic Twist ecosystem comprises the following items.

- The Marketplace. This feature recreates a real marketplace in a virtual space, where users can exhibit, auction and commercialize plastic waste, where it can be monetized with the proposed PlasticTokens.

- The Monetization (PlasticToken). It serves as a support, connecting the other items, and allowing users to reevaluate their plastic in a circular economy.

- Gamification. Plastic Twist was built with gamification by design, delivering rewarding and engaging experiences. It also supports a collaborative gamification mechanism, which is represented in the form of PlasticToken crediting.

- Crowdsourcing tools. Crowdsourcing enables the generation of evolving plastic materials to reuse taxonomy and an open plastic reuse machinery designs repository.

Hence, when users gather their used plastic, they can sell it on the marketplace to the nearest buyer. Once this transaction is completed, the user will receive PlasticTokens, which they will be able to spend to buy revalorised plastic (for instance a set of shirt buttons made from recycled plastic). If the user is not familiar with the plastic ecosystem, they can play with the app to know more, and gain PlasticTokens in the process. If they are more experience and want to recycle plastic in objects that people like, they can use the crowdsourcing tools before selling the repurposed plastic on the marketplace. In a nutshell, these four tools allow for the emergence and sustainability of a circular economy model, hence ensuring the viability of the money.

6.1.5 Outline

This work is organised as follows. First, we give the prerequisites that are required to understand this work. Then, we give more details about the practical implementation of our system, and expose its sustainability via our novel economic model. In this section, we elaborate on the adoption of the ERC20 requirements on a permissioned blockchain, as well as a few other technical points. We also describe how we implemented our cash-in and billing systems. Finally, we conclude our work, while remarking how this work may easily be adapted for many other use cases.

6.2 Prerequisites

6.2.1 Permissioned and Permissionless ledgers

In Bitcoin, the blockchain is permissionless: anyone can join, and anyone can submit a new block without requiring permission from anyone. The only methods to know whether two accounts belong to the same user are if the user acknowledges possession of both accounts; or runs a careful statistical analysis of the exchanges between the two accounts [Mon15]. Naturally, users can choose to identify themselves in a permissionless ledger, but this is not a requirement. A cautious user may very well run two accounts that cannot be linked one to another. A formal definition of permissionless blockchain is that "the identity of the participants is pseudonymous or even anonymous" [Swa15], where participants can be regular users or miners. On the other hand, in a permissioned system, participants are whitelisted (or blacklisted) by an authority, acting as a member provider service. In Hyperledger Fabric [Fou19], the blockchain is permissioned: certificate authorities (CAs) are in charge of determining which users are allowed, or disallowed, in the network. More specifically, in Hyperledger Fabric each organization being part of the consortium is allowed to run its own Member Service Provider (MSP), which is an abstraction containing the organisation's CA. Hence, the accredited CAs are written in the network configuration stored in the blockchain: the genesis block contains a list of accredited CAs. The list can be updated in the future by updating the chaincode, using special transactions.

Since members must obtain authentication from their CA, this implies that all transactions can be linked to the identity of their sender, which is a particularly desirable feature in several applications, such as commercial transactions, where it is possible to verify that a transaction comes from the right place.

6.2.2 ERC20 specifications

ERC20 is a set of specifications that have been designed by the Ethereum community [Woo14], and was released in 2015 [VB15]. These specifications define what a *token* on Ethereum should implement. A token is a virtual manipulable and countable concept, and as such can represent many things, material (e.g. real estate [reg19] or gun ownership [Hes17]), or immaterial (e.g. basic attention [Sof18] or predictions [PK15]).

Tokens are a unit of data that can be distributed to any user of the blockchain network. Their repartition is stored inside a smart contract, which oversees keeping track of the account of all members. Users can interact with the smart contract by sending it money, thus triggering one of the smart contract's functionalities. In Hyperledger Fabric, tokens are natively implemented by the concept of assets. Assets, as the Hyperledger consortium defines it, 'can range from the tangible (real estate and hardware) to the intangible (contracts and intellectual property)', enabling 'the exchange of almost anything with a monetary value over the network, from whole foods to antique cars to currency futures.'

The great advantage of the ERC20 standard is the fact that ERC20 tokens are more reliable for interoperability, as they all implement a set of fundamental functions regarding the tokens.

The ERC20 specifications are quite simple and specify the six functions and two events that must be implemented. The functions are:

- `totalSupply()`, returning the total token supply
- `balanceOf(_owner)`, returning `_owner`'s account balance
- `transfer(_to, _value)`, sending `_value` tokens to the account `_to`, returning a boolean of success/failure
- `approve(_spender, _value)`, allowing the account `_spender` to spend up to `_value` tokens from one's own account, returning a boolean of success/failure
- `transferFrom(_from, _to, _value)`, where an allowed account transfers money from the account they are approved to another account, returning a boolean of success/failure
- `allowance(_owner, _spender)`, returning the amount that `_spender` is still allowed to withdraw from the account `_owner`

And the events:

- `Transfer(_from, _to, _value)`, triggered when a transfer occurs, even when `_value` is zero.
- `Approval(_owner, _spender, _value)`, triggered when the `approve` function is called successfully.

However, in Hyperledger Fabric, there is no such thing as a smart contract, or rather there is only one smart contract, called the chaincode. The chaincode is occasionally updated by the consortium (and not by the users) [ABB$^+$18]. Hence, to be ported on Hyperledger Fabric, ERC20 tokens must be coded by the consortium itself, as they are the only members able to code smart contracts. Furthermore, in Ethereum, in ERC20 tokens are usually obtained by depositing Ether on the related smart contract, which cannot be done natively in Hyperledger Fabric.

Yet, because the chaincode can embed any kind of code, it is possible to add the required logic for an ERC20-compliant token inside Hyperledger Fabric. Because functions of the chaincode can be triggered for no cost by any user (unless specified otherwise by the function), we furthermore obtain a more economical token, as there is no need of sending a commission for each transaction to gain the right to use one's tokens. Note that the costless feature is obtained because in Hyperledger Fabric, it is assumed that nodes have other incentives than mining for running the blockchain. Hence, this property is not characteristic of Hyperledger Fabric, but rather

6.2.3 Preimage resistance

A hash function H is preimage-resistant if, given a hash h, it is hard to find a preimage of h, i.e. a value x such that $H(x) = h$. For instance, assuming that SHA-256 is preimage-resistant, given the hash $h =$`2dd00bd77e0222ced882665481a9c1d9f907309d16e05ed007a1ea63928477a9`, an attacker cannot find a document x whose hash is $SHA256(x) = h$ with better strategy than brute forcing on all possible x.

More formally, we recall Rogaway and Shrimpton's definition [RS04]: let $H : \mathcal{M} \to \mathcal{Y}$ be a hash function, and let m be a number such that $\{0,1\}^m \subseteq \mathcal{M}$. H is m-preimage-resistant if:

$$\forall \mathcal{A}, Pr\left[x \overset{\$}{\leftarrow} \{0,1\}^m; y \leftarrow H(x); \hat{x} \overset{\$}{\leftarrow} \mathcal{A}(y) : H(\hat{x}) = y\right]$$

is not significantly bigger than random. Here, $\mathcal{A}$ denotes a probabilistic polynomial (PPT) adversary, who, given a hash value y, outputs a tentative preimage $\hat{x}$. The affectation $\overset{\$}{\leftarrow}$ means that the selection is randomly uniform amongst the possibilities. In other words, when receiving a random challenge hash y, a PPT attacker $\mathcal{A}$ cannot find a preimage $\hat{x}$ of y with probability significantly better than random.

6.3 PlasticToken Implementation

6.3.1 Assets Implementation in Hyperledger Fabric

In Hyperledger Fabric, the logical elements are coined as assets [Fou19]. One asset can have several properties, and relations can be made among different assets. There can be several instances of the same asset. If a comparison with Object-Oriented Programming were to be made, an asset would be a class. In Hyperledger, assets are stored as key-value pairs: the key being the asset ID, and the value, the asset contents.

The attributes of an asset can be of a variety of types, such as the classical unsigned ints, booleans or string but can also be pointers to another asset, or even a map. A map is a typed collection of `key => value` items and is Hyperledger's implementation of a dictionary.

Note that there is an important conceptual difference between the assets and the tokens that can be granted in an Ethereum smart contract. Assets are native to the blockchain and are the raw material that is exchanged in an Hyperledger blockchain. On the other hand, tokens are not the native unit in Ethereum, rather than an abstraction implemented by a smart contract: in Ethereum, the basic exchange material is Ether; it can be used to activate smart contracts. These smart contracts can then implement an ERC20 token, but Ethers (Ethereum base currency) are not an ERC20 token themselves. Hence, our goal is to implement the ERC20 tokens on a more fundamental level than Ethereum; and we also do this on a fully different blockchain, namely a private one. In a public blockchain such as Ethereum, anyone can code their own ERC20 token; in a consortium blockchain such as Hyperledger, the ERC20 specifications must be embedded into the blockchain source code.

6.3.2 Plastic Twist ERC20

Consequently, we designed our blockchain so that it would be ERC20 compatible. In our implementation, we used one main asset for the support of the ERC20 functions. This asset is called `UserInfos`, is instantiated once per account, and has two attributes:

- `Amount`, an unsigned 64-bit integer (uint64)
- `Allowances`, a `string => uint64` mapping.

The `Amount` property contains the amount of coins on the user's account, and the `Allowances` attribute is a mapping of who can withdraw from this `UserInfos` instance and how much they can. The string key of `Allowances` is a pointer to the authorized user, i.e., a pointer to the allowed `UserInfo` instance.

We observe that an ERC20 token must be stateful, as there are two data that must be saved. Namely, an ERC20 token must save each user's amount of tokens, and the list of allowances that this user has granted to other people. Hence, `UserInfos` contains all the required state storage for a given token, and we are assured that there is no need of another asset.

Then, we implemented the six ERC20 functions in the chaincode. These functions can be called at any moment by any user, and the usual checks are made before validating the transaction. Because these functions are part of the chaincode, and because the blockchain is private (or at least hybrid, i.e. anyone can join, but only after their identity has been verified), there is no need to add a transaction commission; hence the cost associated to the transaction is null. This is a fundamental difference with Ethereum, where all transactions must include a commission fee for the miners, as well as, in most cases, a commission fee for the smart contract itself.

Finally, we also implemented support for the events. Hence, we created a `Events` asset, comprising of four attributes:

- `From`: a string, pointer to the sender `UserInfos`' ID
- `To`: a string, pointer to the recipient `UserInfos`' ID
- `Value`: a uint64, the value of the transfer or of the allowance
- `Details`: a string, either `approval` or `transfer`.

Hence, one can grab all `Approval` events by filtering all `Events(*, *, *, approval)`, and similarly for all `Transfer` events.

The naive implementation of the logging system would be to create one instance of `Event` for each event. However, because the logs are never deleted, this would mean that a node will have to store an indefinitely growing number of logs in its state. Hence, the state of the blockchain would grow linearly with the number of transactions, possibly leading to a memory saturation of small nodes.

For this reason, we implemented the logs so that only one `Event` instance would ever be created on the blockchain. Each time a new event must be logged, the chaincode updates the `Event` instance to contain the new log. Because of the way blockchain works, the previous logs are erased from the state, but not from the history. Hence, anyone can look into past logs very easily, by searching for them in the ledger. Yet, our implementation only requires a constant state size, which is much more scalable for small nodes.

6.3.3 Example of the Internal Workflow

Assume Alice wants to send 10 tokens to Bob, using her allowance on Charlie's account. The following events happen.

1. Alice invokes a `transferFrom` command on the blockchain. The invoke is signed by Alice's private key and associated to her public key. In this case, the `transferFrom` command takes 4 arguments.
   (a) `_from`: Charlie's public key
   (b) `_amount`: the amount of the transaction
   (c) `_to`: Bob's public key
   (d) `_user`: Alice's public key
2. The chaincode calls a function `getTransferFromTx`, which will check that the `_user` variable contains the sender's public key of the sender, as the sender is the only person authorized to add an allowance for someone else on their own account.

3. `getTransferFromTx` parses the amount and the beneficiary, and fails if an error is met (negative amount, non-existent beneficiary).

4. If everything is in order, then the chaincode falls back to the `transferFrom` function. If nothing failed, then this function calls for a change of state of Charlie's account, by calling `changeStateFrom`.

5. `changeStateFrom` verifies that Charlie has enough money on his account and that Alice's allowance allows her to withdraw that much from Charlie's account. If not, an error is raised. If everything is nominal, then `changeStateFrom` removes coins from Charlie's account and terminates.

6. Then, `transferFrom` calls `ChangeStateTo`, which will credit Bob with the corresponding amount of tokens. Finally, the `Event` asset is updated, and is now containing the current `Transfer` event (as a reminder, the `Allowance` event is only triggered when a user allows someone else to use their account, not when someone makes a legitimate call to `transferFrom`)

6.3.4 Transfer Atomicity

As we saw in the previous section, the subtraction of one sender's coin from their wallet and the addition to the receiver's wallet is done in two distinct functions. Then, the obvious concern is the one of atomicity: in case of failure, we must ensure that the net balance is not affected, and ideally, the whole transaction is cancelled. This property is guaranteed by Hyperledger code: when a user launches a transaction (in this case, a `transferFrom` order), either the code is successful, or either the whole transaction is canceled. Hence, as soon as one exception has risen, the whole transaction is canceled, reverting to the before-transaction state.

6.3.5 Rewards and Minting

Contrarily to many blockchains, there is no reward associated with mining. PlasticToken relies on a consortium blockchain, which means the nodes are known in advance and controlled by the consortium. Moreover, in a consortium ledger, the mining process is designed to be easy, and as such there is no real need to reward the investment, as it is minimal. Plus, rewarding miners would lead to a mining competition, thus being less ecological. This is not acceptable as our blockchain promotes a more ecological behaviour.

Hence, new coins are minted via rewards, as a part of Plastic Twist's gamification module: by encouraging users to behave eco-responsibly inside the Plastic Twist ecosystem, they will be rewarded with coins.

Rewards can be automatic (completing a task that triggers a smart contract in the chaincode), authorized (many users are awarded a reward, given by one member of the consortium), or manual (one user signals that they are eligible for a reward). The difference between authorized and manual is light, but relies on the fact that, the user must bring their own proof of eligibility. For instance, rewarding users participating in official events would be authorized: users just need to give their wallet address to the organizer when they arrive to the event. On the other hand, one can design a reward for users who have obtained a given reach on social media. In this case, users must bring a proof that their claim is valid.

This process is summed up in 6.1.

There are, for the moment, four ways of getting rewards. However, this can evolve in the future; if the consortium discovers new opportunities.

- Bringing used plastic to a partner. Users are encouraged to do so, to bring new plastics into the economic cycle. Partners can be fablabs, artists or whoever reuses plastic.
- Answering quizzes and playing games on the gamification app
- Participating in official events
- Days streak on the gamification app

6.4 Additional Features

This section presents two additional features that we added to the Plastic-Token blockchain to make its use as practical as possible.

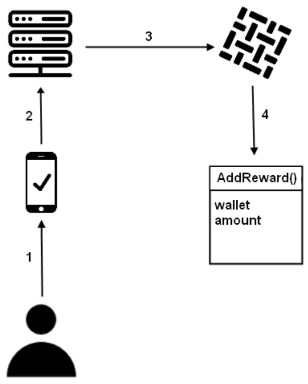
6.4.1 Invoicing and Billing

For business purposes, keeping track of the spendings and acquisition is critical. Especially, businesses must be able to emit invoices, and track whenever they are paid.

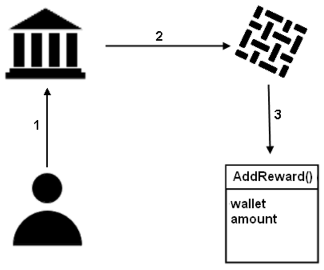
For this, we used two new assets, `Bill` and `Item`. `Bill` contains a list of items and an owner ID (the ID of the bill issuer). Similarly, `Item` contains a description of the product, the amount of products bought, and its unitary value. This relation is summed up in 6.2.

For instance, let us assume that user U wants to buy 3 items (with different multiplicity) to the seller S. U contacts S and lets them know about their inquiry. Then, S generates three `Item` descriptions (one for each item i_1, i_2, i_3, with their respective multiplicity), and one `Bill` listing these three items. The bill is then published on the blockchain, and added to S emitted bills in their `UserInfos`.

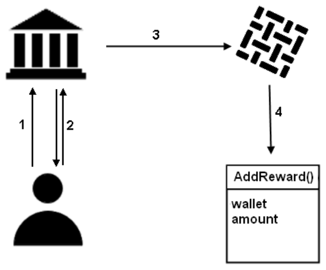
If U agrees to the bill and pays the bill, they emit a special transaction, `payBill`. The function takes only one argument, the bill ID. Then, the chaincode retrieves the corresponding bill, and the bill's owner ID.



(a) Automatic rewarding. The user completes an action on the app (1), which relays it to the app server via a dedicated API (2). Then, the server triggers the reward smart contract (3 and 4). Only allowed servers are allowed to do so.



(b) Authorized rewarding. Organizers scan the address of eligible users (1), and trigger the reward smart contract (2 and 3).



(c) Manual rewarding. User go to an organizer to claim their reward (1), then provide required proofs (2). When the organizer is satisfied, they trigger the reward smart contract (3 and 4).

Figure 6.1 – Different rewarding modes

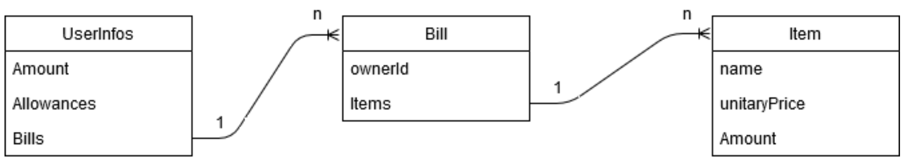


Figure 6.2 – Entity relations for billing

Then, the chaincode computes the required price to pay. For example, assume that the bill contains three items $item_1$, ..., $item_3$, with multiplicity m_i and unitary price p_i for each item. Then, the chaincode computes $\sum_i m_i \times p_i$ as the bill's price.

Finally, having retrieved the owner ID and the total price, the chaincode calls the ERC20 function `transfer(owner, total price)` on behalf of the payer. If the call succeeds, then the bill is paid, and deleted from the state. Otherwise, the bill remains unchanged, and the transaction fails. These steps are schematized in 6.3.

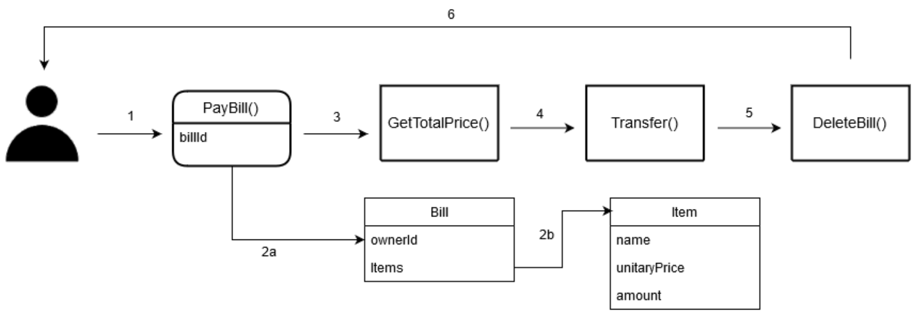


Figure 6.3 – Chaincode steps for paying a bill. 1: user invokes the payBill function. 2: the function retrieves the associated bill and its owner, and the items linked to it. 3: The price is computed. 4: the chaincode tries to send the total price from the user to the bill owner. 5: if successful, the bill is deleted, and 6: the user receives a confirmation.

Furthermore, the asset `UserInfos` is upgraded, and contains a new property: `bills`, which is a list of pointers to `Bill` instances: they are the bills generated by a user.

Additional security can be added by limiting a bill payment to the user it was issued to. While the adaptation is easy, it is left as future work.

6.4.2 Physical Coins for Punctual Events

Even though PlasticToken is a virtual currency, used for plastic reuse, there is a need to print physical tokens. For instance, during dissemination events, it may be useful to have physical tokens so that anyone can use the currency even if they do not have downloaded the app. Furthermore, beta-testing showed us that people were more likely to interact with the Plastic Twist ecosystem if they could at first use physical tokens. Hence, we devised a way to issue physical tokens in a punctual event securely: the physical tokens only have a limited period, and can only be traded inside the event. We now implement the scheme, which is quite simple and relies on one additional asset.

We hence added the asset `CashVoucher`, which consists of two attributes:

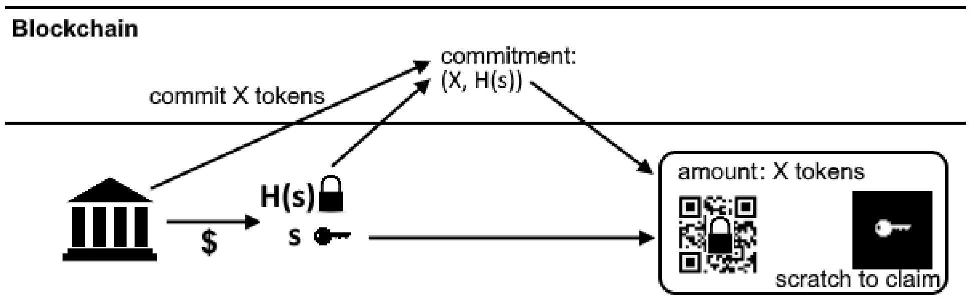
- `Amount`, the number of tokens committed in the voucher
- `Hash`, a preimage-resistant hash of a secret value.

For security reasons, only members of the blockchain consortium can issue physical tokens, and they only can do so by locking tokens they own on the blockchain: for each coin they physically print, they lock the corresponding amount on the blockchain. Hence, any user converting their tokens back to virtual will receive these committed tokens.

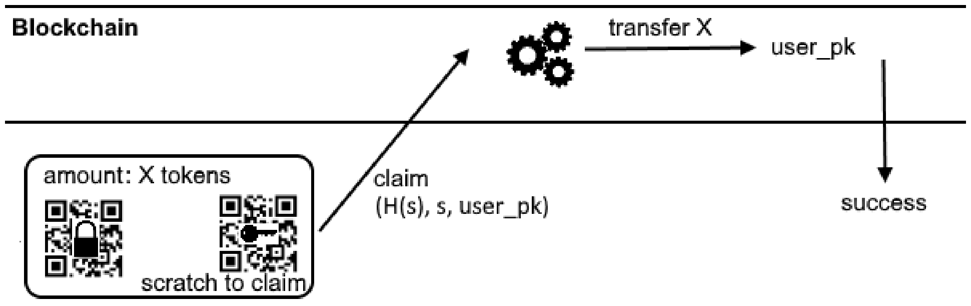
Hence, our protocol is as follows.

1. **Commitment**. When an organizer is preparing an event where they need physical tokens, they first commit some PlasticTokens which will be removed from the organizer's balance, and place them in a `CashVoucher`. While creating the `CashVoucher`, the organiser also selects a random 256 bits secret s, and discloses $H(s)$, where H is a preimage-resistant hash function such as SHA-256. Hence, the `CashVoucher` instance contains both the committed value and the hash of the secret.
2. **Printing**. The secret s is printed, in the form of a QR code, on the physical coin, and hidden behind a scratch card. The scratch cards can be easily printed by organizers, with a minimal amount of material (for instance, the DYI tutorial in [Fis17] only requires paint and dish soap). The scratch card also contains the hash, and the number of coins they are equivalent to, and can be traded by anyone. Once printed, the secret s is discarded by the organizer.
3. **Use**. When receiving a coin, a user might fear that they are not linked to a real commitment. Hence, by scanning the QR code of the hash, they can verify that there is indeed a commitment of the correct amount of tokens mapping to their hash. Furthermore, this scanning will prove to the user that the physical coin has not been claimed yet. A rational user should always scan the coins they receive; however, given that they trust the organizer, and forging a malicious scratch card in a limited amount of time (usually, such events last half a day or one day) is tedious, they can decide not to.
4. **Reclaiming**. At the end of the event, or whenever the user wants, they scratch the tickets they own and retrieve the associated QR code: the QR code contains the secret value s linked to the physical token. By claiming s to the related `CashVoucher`, the user retrieves the associated committed virtual coins in their wallet.

The process is summed up in 6.4. This scheme was already used by the authors in another work describing the economic aspects of PlasticToken.



(a) Cash-in mechanism. Secrets are randomly generated by the organizer, and the secret s is discarded by the issuer after the card has been printed.



(b) Cash-out mechanism. If the secret has already been reclaimed, an error is returned.

Figure 6.4 – 6.4a: How organizations print tickets. 6.4b: how individuals claim their coins after scratching the secret key, at the end of the event. Additional checks may be added to the cash-out mechanism.

Note that the user must trust the organizer in this context. As the organizer is the one printing the tickets, they have access to all secrets. Furthermore, they can also print several tickets linked to the same secret, or print tickets not linked to any secret. While the user can detect the last event, the previous one is more difficult to detect (as the user must own the two tickets linked to the same commitment). Finally, there is no guarantee that the organizer will effectively discard the secrets and not use them at the end of the event to claim all the committed tokens. However, organizers can be trusted on this fact, as they are the ones running the blockchain. Because one already has to trust the consortium to run the blockchain smoothly, they can also trust them when organising events.

We now give a brief proof of the system's security, assuming the organizer is honest (which can be assumed, given the discussion above).

6.4.2.0.1 Security against coin stealing To steal a coin, one must claim a commitment, whose associated physical token they do not own. Hence, we assume that the attacker is trying to guess the secret s linked to a hash $H(s)$, without knowing s.Since the hash function we use is preimage resistant, there is no better solution than to brute force all possible s, i.e., to try all possible 2^{256} bits secrets. This attack would lead to a computational cost of 256 bits, way above the recommended NIST standard of 112 bits of security [BBB+07]. However, an attacker might not want to attack a specific token, but rather any possible token (this is called an existential attack). In this context, if the hash function is preimage-resistant, the best-known attack relies on the birthday paradox, where the attacker finds a collision after about 2^{128} tries, which is still bigger than the security standards.

6.4.2.0.2 Security against malicious forging Another attacker might want to forge their coins, so that an unsuspecting user will believe they hold a valid coin. There are two ways to do so. The first naive one would be to choose a random secret s', and print tickets containing the hash $H(s')$. However, a user might scan this ticket and realise it is invalid (as an attacker cannot create `CashVoucher`s on the blockchain). However, a more subtle attack can be carried. Remember that the `CashVoucher` are publicly available on the blockchain; hence, the attacker learns which hashes have been used for this event. Then, they can print tokens just like the organizer did, but with a fake secret under the scratch section. The user will only realise they have been fooled much later after exchanging with the attacker, when they want to claim the virtual tokens back. While this attack seems doable in theory, in practice we observe that events are punctual and that attackers must exactly mimic the legitimate physical coins: type of paper, design, display, type of scratch surface... Because events are limited in time, it is not feasible for an attacker to carry the attack in such a limited time slot. We observe that this security in practice is enough, as school fairs and festivals rely on a similar assumption-that no one has the time to mimic the physical tokens in a small amount of time.

6.5 Conclusion

In this work, we have presented the first system implementing ERC20 standards on a consortium blockchain, Hyperledger Fabric. This system offers a novel currency, the PlasticTokens, which enables a local and circular economy for its user, in which everyone is encouraged to reduce their plastic waste. The rewards incentivise this policy. The rewards are both an incentive for the user to adopt the desired behaviour, and a natural way of dealing with the natural growth of the system, as users will progressively subscribe. Furthermore, the marketplace, another piece of the Plastic Twist project, fosters the circular economy, thus ensuring the users will be able to use the PlasticTokens in an intended way.

We also introduced a way of issuing bills and paying them. We also introduced printing physical coins, with which users can physically trade PlasticTokens for goods and services. Even though the implementation, by nature, cannot be fully secured, it is secure enough to be used with sufficient trust in a real-life event lasting one day, which is the maximum event duration the Plastic Twist pilots have reported.

As this model has been specifically designed for reducing plastic waste and incentivising plastic reuse, it is not immediately adaptable to other business models. However, given that the code is not related to plastic in any way, it is trivial to adapt our scheme to other circular economies to reduce waste.

Finally, the whole source code is open-source and publicly available on Github. The link will be available after peer review.

Chapter 7

Enabling trust in healthcare data exchange with a federated blockchain-based architecture

Contents

We propose a new healthcare data exchange platform for research centers, hospitals and healthcare institutions. Our model is based on a federated blockchain network that interconnects the healthcare institutions and orchestrates the data life cycle from the data publication to the data consumption. The blockchain is responsible for keeping the traceability of the whole process and we use a specially designed smart contract to control the data sharing process. Moreover, we provide the means to enforce GDPR and thus achieve a GDPR compliant model.

7.1 Introduction

In the health industry, it is of utmost importance to constantly research about different topics. In this sense, the development of new techniques and treatments on diseases and epidemics strongly relies on gathering data from multiple sources and populations.

However, gathering data for research purposes can be an exhausting and daunting job for researchers. Obtaining the correct data and of good quality has high monetary costs and it is time-consuming. Moreover, data gathering usually must also overcome ethical and methodological challenges, which further complicate the scenario.

It is crucial then to provide the means to lower the costs and fasten the process in a secure way in order to allow data exchange between research institutions and medical organizations. However, data exchange brings a wider range of challenges to the table.

From a socio-technical perspective, current data protection regulations and standards apush forward more rigorous data management approaches. Regulations such as the General Data Protection Regulation (GDPR) and the Health Insurance Portability and Accountability Act of 1996 (HIPAA) increases the responsibilities and accountability of data controllers and processors when processing health data. For instance, under Article 9 of the GDPR, "data concerning health" is defined as a special category, and thus must be further protected than normal personal identifiable information [Par16] which establish further responsibilities for the processing of health-related data. Thus, exchanging data between actors cannot be tackled with a straightforward approach.

On the other hand, by not having a system that is properly designed system for data management (particularly for data exchange), the business can suffer huge consequences. For example, the average cost of a data breach per capita in the healthcare sector is of 408 USD [Pon18]. Given that data in the movement has a higher risk of being breached, this is indeed an important number to consider. In addition, patients and data subjects have certain expectations about the management of their health data. Thus, in the case of a data breaches, organisations that have suffered from them, can also see their reputation and trust affected.

As a consequence of the above situation, gathering consent from the patients to use their personal data has become more difficult. Therefore, healthcare institutions need to improve their internal management processes and in particular those related to data and consent management to overcome this problem.

All in all, we have designed a model that can help in this direction, and that would allow healthcare institutions to have access to data in a faster whilst also sharing the data in a secure and trusted manner. Our design seeks to make the data exchange process faster between the actors without losing crucial properties like traceability and accountability. By improving the whole process, our design contributes to a better data governance within the healthcare data exchange. This work aims to explain our model, its design and architectural decisions emphasizing on how they contribute to achieving such goals.

7.2 Related work

In [BKMPK18], the authors propose a blockchain model that ensures GDPR compliance and provides the means to carry out sensitive data sharing between network participants. This work generalizes work from [BKMPK18] by clearly defining a compatible architecture and detailing its main components. As a result we obtain a specification that has been instantiated in the Horizon 2020 project MyHealthMyData[Con16a] but also that is of it own interest to study.

To the best of our knowledge no other blockchain-based model for similar purposes can ensure GDPR compliance as our model can. However, for instance, Medicalchain [Med19] which uses blockchain technology to store health records does not provide information on whether or not their solution is GDPR compliant. The only reference to the GDPR in their whitepaper, when referring to data structures that are being used, states that: “These are subject to change depending upon different regulations and requirements to make the Medicalchain platform HIPPA and GDPR compliant”.

7.3 On the applicability of a federated blockchain

This section we will argue on the applicability of using a federated blockchain as the backbone of our model to enable trust in healthcare data exchange. We will first introduce the main ideas behind a blockchain.

A blockchain is a type of data structure that is replicated, shared, and synchronised between different participants that rely on a peer-to-peer network to connect with each other.

As a data structure, roughly speaking, it is an append-only ledger organised as a chain of blocks where any modification in a block compels the regeneration of the following blocks in the chain, since the blocks in the chain link to its predecessor by a hash pointer. In addition, blockchains usually work with blocks timestamps to make it more difficult for an adversary to modify the chain. By following this design, the modification or deletion of any block can be extremely difficult to achieve.

It is replicated because participants hold their own local copy and it is synchronized because participants rely on a consensus mechanism to govern the management, updates and operations on the blockchain. As a result, parties can agree to a specific state of the blockchain as the valid one in a distributed manner (with no single entity in control).

With the above in mind, the following properties should be ensured.

- **Persistence [GKL15]:** Once an honest party reports a transaction "deep enough"[1] in the ledger, then all other honest players will report it indefinitely whenever they are asked, and at exactly the same position in the ledger. This property is usually referred as the immutability of the blockchain.

- **Liveness [GKL15]:** Transactions from honest parties will be included in the ledger of honest parties. This property roughly states that the blockchain will be able to process transactions coming from honest users.

It is worth clarifying what an honest party means in this context. To do so, let us introduce a distinctive feature of a blockchain regarding the management of participants in its network.

- **Permissionless:** Usually the most well-known type of blockchains, where anyone can join and participate. The most famous examples are *Bitcoin* and *Ethereum*. In these blockchains no assumptions should be made at all regarding the honest behaviour of the parties. This means that some parties may arbitrarily decide not to follow the protocol in different ways to take advantage. In these scenarios, mechanisms to reward honest behaviour need to be triggered as a way to discourage malicious users from deviating from the protocol. This comes with a cost that can be reflected as a considerable overhead to the system.

- **Permissioned:** Participants in these blockchains can be fully identifiable and thus access can be granted or denied to them. Also, since the

[1]If a transaction **tx** is committed in the block $n + k$ that is contained in a blockchain of $n + k + t$ blocks, t can be seen as the deep of the transaction **tx** with respect to the sate of the blockchain.

participants can be identified, permissioned blockchains can use different consensus protocols that cannot be used in the permissionless setting and benefit from it.

As the reader may find different terminology in the literature, such as public or private blockchains, we would like to clarify a relevant aspect related to it. Public blockchains usually refer to permissionless blockchains, whereas private may not be the analogous (permissioned). We consider private blockchains as those that are manage privately within a given organization and thus prefer the term federated to talk about permissioned blockchains that are not under the control of a single entity.

Thus, when we talk about federated blockchains, we refer to the fact that the permissioned blockchain in question is not governed by a single entity but rather by a federation or consortium of organizations. These organizations, in agreement, define the policies for its access control layer which will define which permissions are granted to which concerning the blockchain.

It is also worth mentioning that the fact that participants in a permissioned blockchain can be fully identified does not guarantee that all of them will behave honestly. It says, at best, that there will be fewer incentives for them in deviating from the protocol (as they can be caught if doing so).

In some scenarios where parties that are known to each other need to cooperate or work together, but do not fully trust each other, a federated blockchain can help. Moreover, when cooperation needs to follow a business logic that cannot be afforded by the parties cannot afford to be public, a federated blockchain may be more suitable.

The above scenario can be applied to sensible data exchange if we think in a consortium or federation of organizations that are willing to cooperate with each other but at the same time do not trust or cannot afford the cost to trust in a single entity.

As an example we can mention projects like Hyperledger Fabric[oSN15] which provide different tools within the Hyperledger framework to build and deploy enterprise blockchain solutions for similar cases. Moreover, Hyperledger Fabric works based on pluggable-consensus algorithms that can be plug in and out accordingly to the needs of the solution.

7.4 Overall architecture goals

In section III we presented the applicability of a federated blockchain in allowing participants of a network to agree and enforce a shared business logic to rule the healthcare data exchange process. In this section we will specify the main goals of our model, taking the federated blockchain as a starting point to define such goals.

7.4.1 Process automation

As stated in Section 1, a key aspect of being addressed is how to fasten the process of the data exchange in a secure way. In addition, smart contracts can be helpful when it comes to process automation, as we will discuss below.

First, let us recall that the idea of smart contracts is not new. Szabo originally proposed in 1996 [Sza96] to allow two or more parties to agree on a subject by delegating the management and enforcement of a contract to an application. Nonetheless, it was not until the rise of blockchain and decentralized applications that smart contract development flourished.

In 2015 Ethereum achieved for the first time the execution of Turing-complete code using a blockchain to that end. This, in turn, made the original idea of smart contracts possible in terms of "autonomy" and gave rise to what we now call decentralized applications.

In brief, the main idea behind smart contracts is to program rules and business logic to be later on enforced independently of the participants' interest. So once a smart-contract is deployed, it will run and take actions solely based on the code that has been written to it. Ideally, under this paradigm, participants in the network would invoke a given smart contract by providing input and retrieving the output but would not be able to interfere in the process. A party would not be able to convince another party to accept an output that is different from the one defined by the smart contract logic given a fixed input. It is worth mentioning that this poses challenges when dealing with non-deterministic computations. We refer the reader to [ABB+18] to further look into the perils of such a challenge.

With the above in mind, smart contracts are very appealing when it comes to defining different business logic among participants that may have a conflict of interest or that may try to take advantage of other participants who also rely on the same logic.

Accordingly, the Hyperledger fabric documentation, “smart contracts are not only a key mechanism for encapsulating information and keeping it simple across the network, but they can also be written to allow participants to execute certain aspects of transactions automatically.” [oSN15]

Since the smart contract will “play by the rules by default”, participants can trust their execution and thus, all the parties can trust process automation. This is what drives our first goal: defining in a secure and trustful manner a process automation to handle the data exchange process between parties. We will explain in the next sections how this can be achieved by using a smart contract.

7.4.2 Data traceability

We refer to data traceability as the process to know what type of data has been in the system, when and who used it, and under which conditions/purposes.

This idea of data traceability helps to improve the model of data governance. Overall, we can understand data governance as the data management and IT strategy by which organizations establish rules, policies, models and management of the data [OBL16] [CNS15]. More specifically, NIST has adopted the following definition for data governance:

"A set of processes that ensures that data assets are formally managed throughout the enterprise. A data governance model establishes authority and management and decision making parameters related to the data produced or managed by the enterprise" [CNS15] [NT15].

To enable a proper data governance model, it is important to control data throughout its lifecycle. This implies, among other things, that data owners should know where the data is, who has used it, when and under which conditions. This item is particularly important for data exchange or sharing. In addition, organisations should record to whom data has been shared to comply with regulations and enable data subject's rights and security measures. Thus, having a good set of policies, rules, frameworks and systems that allows data owners to trace data, strength the data governance.

To accomplish this task, we introduce the following concepts:

- **Proof of existence**: It should be clear for any given data to be exchanged what is the exact content and under which conditions it can be shared. For healthcare data, the relation between the data and the consent given to its usage need to be defined at the very early stage of the process. Thus, it is important to be able to prove the existence of such relation at any given time. This, in turn, helps to know what type of data has been in the system.

- **Proof of matching**: To better provide the means to the when and who used which data, we introduce the notion of proof of matching to capture which and when the data has been used and by who. Proof of matching then implies that the party sharing the data can prove no only when that data has been used but also by whom.

7.4.3 Decentralization

One of the biggest challenges of data governance in a centralized way is that a single entity carries out data the management. Therefore, there is a risk that records can be tampered with or altered, either in a malicious or unintended way. This entails that data subjects must rely on the organisation's policies and trust that they are being enforced. A direct consequence of this model is that costs tend to be high due to all the work that needs to be done by the entity in charge of the data management.

Also, data controllers may be reluctant to delegate the whole process of data exchange to a single entity even though they could afford the cost of it.

Decentralization in this scenario allows every party to keep the control of their own data and to participate in the decision-making process of data exchange. Moreover, by distributing the responsibility of the process among parties, there will be fewer chances for the information to be tampered with if most of the participants remain honest.

7.4.4 Auditability and GDPR compliance

Auditability is a key component in data security which in turn can help to achieve GDPR compliance. Given the traceability feature of the system, it is possible to examine and systematically review the data management. Even more, given the design of the network, it is possible to achieve the desired granularity in the retrievable process, with the possibility of knowing specifically where each data set has gone.

Due to the immutability principle of the blockchain, it is possible to audit the data trails and data management with assurance that the database has not been tampered with.

One of the interesting aspects of this design, is that the "right to access" [Par16] enabled by the GDPR could be accomplished by the model. If a given user wants to carry out its own right of knowing where its data has been and with whom it was shared with, a positive answer should be delivered to him. This is what we seek by setting this goal, to take the immutability and decentralized properties of the system to ensure that every node in the network can be able to trace its own data. Moreover, some other rights such as the right to be forgotten should also be subject to exercise in order to be fully GDPR compliant.

7.4.5 Enable trust

When designing such a system, one major concern is to provide a secure way to transfer and exchange data between the parties. In this sense, to comply with data protection regulations and possible malicious actors, it is crucial to preserve the confidentiality and integrity of the data. The model should also assure that entities are authenticated and part of the network to clearly identify every action.

The properties above of the system are important for data exchange for three main issues: data protection, regulatory compliance, and enabling trust. The first two, data protection and regulatory compliance, are tightly related, as we seen before when referring to the GDPR.

Furthermore, it is also important for any organisation handling personal data to trust their data subjects. Given that health data is considered a sensitive (or special category), special attention must be given to security when data is in movement. If a data breach occurs, as it has been noted before, reputational harm can be done to the organisation. Thus, our goal in this sense is to prevent as much as possible data leakages, which is also another reason for enhancing data traceability.

7.5 Our model

Even though the federated blockchain is a central component in our model, it is certainly not the only one. Therefore, in this section, we will dive into each of the components explaining, their main functionalities and how these contribute to achieving the previously defined goals.

Let us first start by saying that under this model different organizations agree on participating together in the same network. As an example, one can imagine a network composed of two research centers and one hospital.

Following this example, every organization will have their own users and every user will be recognized as a valid one by other organizations. This means that organizations have control over their own users and any other network member recognizes credentials issued to them.

It is important to highlight that every organization runs at least one node in the blockchain network and thus the governance of the federated blockchain is distributed among the organizations.

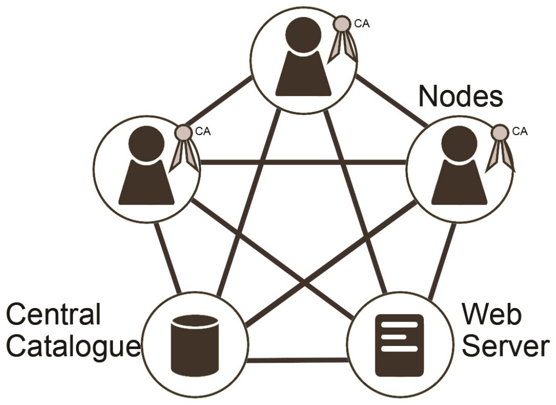


Figure 7.1 – Model representation

7.5.1 Central web server

We define a central web server as the users' endpoint to communicate and interact with the whole system. A user belonging to one of the organizations will use this web server to carry out all its requests and interaction with the network.

It is worth noticing that introducing a component of centralization in this architecture does not prevent the goals we previously defined related to the decentralization since the web server will be an access point but will not hold any sensitive data (other than information related to users authentication).

The idea of the central web server is to facilitate the interaction with the system and with the central catalogue (which will be introduced next).

Since we have a federated blockchain network, authentication needs to be handled differently.Therefore, we propose a single sign-on approach where users authenticate with the web server specifying username, password and organization so that the web server can validate these credentials with the affiliated organization. This, in turn, means that if one of the organizations experiences problems, only users from that organization will use the system.

7.5.2 Catalogues

To create a data request a user needs to know which data is available in the system to ask for it. To manage the data available in the system, we use of local catalogues held at every data controller and one central catalogue.

Local catalogues index the data that will be available to the network from the data sources. This indexation process includes the normalization of the data (the generation of metadata from the actual data) and the registration of the metadata in the central catalogue.

The central catalogue is just an aggregated version of the available data, enabling users in the system to browse for data in a consolidated way. We rely on metadata to describe the available data to avoid direct links to the actual data. We should stress that is also desirable to require data controllers to perform at least one level of anonymization to the data before sharing it.

This way, when a user browses the metadata from the central catalogue it will know which type of data is available but not who is providing it.

7.5.3 Certificate authorities

From the authentication point of view, every organization has its own Certificate Authority with an API (CA API) and a second API (Authentication API) that is integrated with the web server to handle the authentication with the users. The web server consumes this Authentication API to carry on the single sign-on process and, in turn, will consume the CA API to perform the right mapping between the user's credentials and associated public keys. This model can be easily deployed with Hyperledger Fabric for example.

By managing certificate authorities we can authenticate any action performed in the system. As for the blockchain, every CA will generate certificates for its users and for the TLS communication between the nodes.

7.5.4 Network nodes

We define a network node by the following components:

- A blockchain peer, eventually with more than one peer running in the same node for performance purposes.

- A driver (backend application) which implements the main business logic. It processes requests from the central web server and communicates with the local peers to execute and process transactions.

- A local mapping database that is used to track of the references that every data item has in the blockchain. It links a data item with every data request in which it was used.

- A certificate authority (CA) responsible for issuing certificates within the organization.

- An authentication API that communicates internally with the CA and externally with the central web server. The idea behind this is that the authentication API connects to the CA to generate a user and store it locally. It then provides the means to the central web server to authenticate users following the single sign-on approach.

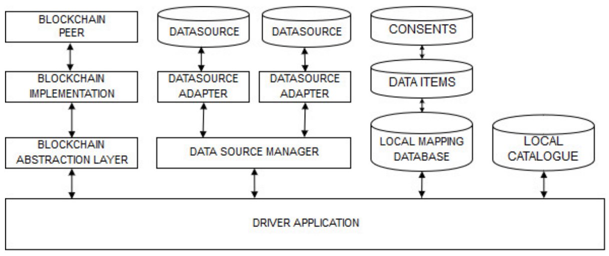


Figure 7.2 – Logic abstraction of the driver

Two operation modes for the driver are defined, with data sources and without data sources. Organizations can process and request data, but not necessarily every organization on the network will process data. This means that some organizations may will only ask for data but never share data. This is the case with pharmaceuticals, for example. In this case, as the organization will not be sharing data, the driver does not need to communicate with a local catalogue or data sources adapters. Depending on the driver's operation mode, a network node may have other services running or not.

7.6 Data exchange flow

Our defined goal on process automation introduced the idea of delegating the administration of the data exchange flow to a smart contract. This section we lists the steps that we defined for the data exchange flow and how they interact with the logic established by a smart contract that we called ExchangeFlow.

Like a smart contract, ExchangeFlow provides the following methods that can be invoked and manage different assets.

- createDataRequest: This allows the user to define a new asset in the system and record information on the blockchain regarding a new data request.
- query: allows the user to query on the assets which exist on the system.
- registerData: This allows the user to define a new asset in the system and record information on the blockchain regarding the new registration of data.
- registerResponse: allows the user to define a new asset that contains a response for a data request that was previously defined in the system. T. The data requestor later retrieves these assets to process the request.
- updateDataRequest: This allows the user to update a data request asset in order to reflect whether or not all the responses have been sent. This facilitates the processing of closing the request and delivering the results to the end-user.

To better explain the details of the step by step process, let us first stress the following related to the actual way in which the exchange is performed. A session key is generated every time that data items are shared. This session key is used to encrypt those data items, so it needs to be shared with the data requester. To share the session key, a public key (which we introduce in step 2) is being used to encrypt it so that only the data requester can obtain it. The data requester will download the data items (which are encrypted) and will have to decrypt them with the corresponding session key.

We detail below the step by step process, which consists of the following steps: 1) Data registration, 2) Data request, 3) Data fetch, and 4) Data response.

7.6.1 Data registration

The driver is used to publish data to the network. This process, called data registration, consists of several steps that we will describe in detail.

Recall that a driver has its own local catalogue and data sources adapters. The adapters are used to fetch the data from the data sources. As data can be heterogeneous and thus come from different sources, we define such adapters to achieve a modular design. Every datasource adapter implements the same interface. Thus, adding a new data source boils down to implementing the datasource adapter interface for that datasource.

Different datasource adapters are handle ny a datasource manager who communicates back and forth with the driver application.

When a datasource receives new data, its datasource adapter communicates with the datasource manager, which in turn communicates with the local catalogue to index the data.

Once data is indexed, it needs to be registered so it is available to the network.

To register the data, the driver invokes ExchangeFlow.registerData, which in turns creates an asset attesting the registration of the data in the blockchain. This works as a **proof of existence** and allows the driver to prove later that it had available data with a given consent by the time of the registration.

Note that the consent must be specified to register the data, but most importantly the consent will need to be checked before sharing data. In this sense, the consent management is the process of checking that these two steps are being done correctly (i.e. a consent is defined and properly checked for every data item).

To register the data, the local mapping database is used. The content of this database are pairs (key,value) where the key is a unique reference for every data item, and value is a set of references from that data item to the blockchain. In the value field, a tuple (blockchainTransactionId, offset, consent, hash,hmac) is saved every time a data item is registered with a consent. Below we explain the different fields for the tuples in value:

- blockchainTransactionId: indicates the transaction in the blockchain where the data item has been registered.
- offset: is an integer assigned during the invoke of ExchangeFlow.registerData. A global counter is used to assign a unique value to every data item.
- hash: is the cryptographic hash of the data item to make sure that the data has not changed.
- hmac: is a key-hash message authentication code of the data item hash, its unique id and the consent. This allows the driver to prove data integrity and authentication for the data item and its consent.

It is important to recall that such mapping is entirely in the driver of the data controller and thus only that party, which is the intended one, would be able to respond in case of an audit. For the rest of the parties, the information recorded on the blockchain will be useless without the references at the local mapping.

7.6.2 Data request

We now describe the data request process. First; a user logged into the system can use the central web server to browse data available from the central catalogue. At this point the user knows which type of data is available for request but does not know who has such data. Thus, the user knows which types of consents are compatible for the available data as this was previously specified during the data registration process.

Once that the user-defined the type of data that he will request, the next step is to communicate such a decision to the network. At this point, the user-specified how he is going to use the data and for which purposes. Both need to be compatible with the consents defined by the data registrants; otherwise, they will not carry out the exchange.

With the above information the central web server communicates to the driver who belongs to the user organization and forwards the request.

The driver processes the request by invoking ExchangeFlow.dataRequest specifying the following information:

- dataRequestId: a unique id assigned globally to every dataRequest when before creating the asset.
- dataRequestInfo: general information which includes the purpose of the request as well as the requested consent.
- publicKey: a public key is created per data request to allow all the entities to communicate with the requester when sending their responses privately.
- status: a field indicating the status of the data request that will be updated once that all the responses have been collected. This is field is also used to notify the user in case if problems arise from the data exchange process.

Finally, a corresponding transaction reflecting these changes is added to the blockchain. This will allow all the networks nodes to be notified of a new data request when reading the updates from the blockchain.

7.6.3 Data fetch

Upon notification of a data request, every driver checks internally if it has available data to exchange. Then, as the data request specifies the conditions on the data usage and the required consent, every driver can check and decide weather to share the data.

If the answer is positive, the data needs to be fetched from the data sources, and a data package needs to be built from the available data. We call this process data fetch.

7.6.4 Data response

Data responses can be positive or negative. To respond, an invocation to ExchangeFlow.registerResponse must be made by specifying the following fields to create the corresponding asset:

- dataRequestId: the id of the data request it is responding to.
- result: indicates if it is a positive answer or a negative answer. If it is negative, the rest of the fields can be ignored.
- info: indicates general information on the response. For example, the hashes of the data items to be shared so they can be checked afterwards.
- privateField: a session key to decrypt the data items being exchanged is encrypted with the publicKey obtained from the data request and sent in this field.
- downloadLink: a link to download the data items.
- bitmapHash: the hash of the bitmap for the resultset (set of data items to deliver). A data request would normally use fewer data items than the ones that the driver registered. The driver calculates and stores a compressed bitmap from the offset of the data items involved in the request-response. Thus, bitmaps are used to establish the link between data items that a data request has used for every data request. As a **proof of matching** the driver will save the bitmap hash. This allows it later to prove that a given set of data items have been used in a given data request. It is worth noticing that since the bitmaps are stored locally on each driver only the driver who stores the bitmap can know where to check where the data items have been used.

It is important to notice that the actual data exchange is carried off-chain and no direct information on the actual data is saved on the blockchain.

The data requestor will read the responses from the blockchain and process them one by one downloading the data sets and decrypting them with the corresponding session key. Finally it will invoke ExchangeFlow.updateDataRequest to end the data request.

Information related to the whole process can be later checked and retrieved by invoking ExchangeFlow.query on the corresponding assets.

7.7 Conclusions and further work

In this work, we presented a novel approach to empower organizations in the decision-making process of exchanging healthcare data while avoiding the need to rely on a third party. As said before, this can be expensive from the monetary and data breaches point of views. Under this model, every institution is responsible for its own data and the actions it takes with it and thus can apply its own policies. Moreover, they can easily trace to whom they deliver the data and under which conditions. From the data requester perspective, they are forced to declare in advance such conditions, so that the exchange rules remain transparent for the involved parties and auditing the process becomes easier.

Process automation in this sense makes it also easier to gather data faster, which in turns can boost its usage in fields like health data analysis and mining.

Furthermore, our model is the first GDPR compliant blockchain-based solution in the health data exchange domain to the best of our knowledge.

As future work, integration with novel approaches to enhance privacy (e.g developing tprivacy preserving techniques in smart contracts, secure multi-party computation and zero-knowledge) would allow the extension of this model to different domains.

Conclusions

Summary of the contributions

The research work carried out in this thesis covers various subjects around cryptography and blockchain, including theoretical and practical results on privacy-preserving user identity protocols, software obfuscation, and a cryptocurrency scheme for new economic models.

This thesis is organised into three parts. The first part covers a brief story about blockchain and the main concepts of modern cryptography. The second part includes the theoretical results on privacy-preserving consensus, electronic voting on blockchain and control-flow graph obfuscation for software protection (i.e. intellectual property). Finally, the third part includes the applied research results proposing a management model on blockchain for GDPR compliance and a new cryptocurrency scheme for a circular economy.

Privacy-Preserving protocols: Chapter 3 presents a new privacy-preserving user identity consensus protocol using blind signatures schemes for transactions unlinkability in the permissionless blockchain. The construction proposed is based on a BFT algorithm for the transactions validations and for agreeing on the new block. The protocol is designed based on a generalised construction of blind signatures to allow users to issue a blinded signature to keep their identity private. Therefore, for practical uses, the protocol can be implemented with different blind signature schemes like Okamoto-Schnorr blind signature [Oka92], Chum's blind signature [Cha82], Fuchsbauer et al. [FHS15], among others. This protocol achieves transaction unlinkability (using a blind signature scheme), consistency and liveness, making it suitable for multiple blockchain implementations. Moreover, all the components used in this protocol already exist in Permissioned or Private Blockchain architecture. Hence, our algorithm performs the signature blinding process by using the membership authorities or the user administrators that already exist in a Permissioned scheme, making it efficient for these kinds of Blockchain architectures.

In Chapter 4 it is proposed a new decentralised electronic voting scheme. This protocol is called DABSTERS, and uses a new architecture to enhance the security level in eVoting and ensure the trustworthiness needed in any electoral process. DABSTERS design uses private Blockchains with a new privacy-preserving consensus algorithm based on Okamoto-schnorr blind signature to guarantee vote integrity and voter's privacy due to the unlinkability property that the blinded signature has.

Chapter 7 covers an applied research result on blockchain and privacy-preserving regarding the European General Data Protection Regulation (GDPR). This work presents a novel approach for decentralised data governance in healthcare data exchange. Each institution is a data controller, and it is responsible for all the actions over the data and the data policies for data exchange or data processing. The solution allows the data controller to communicate with the other institutions through a federated blockchain to demand access to pseudo-anonymised. Then, the system checks the consent given by the data owner to each data controller. If the consent allows the data exchange, the system will make available the data package, and it will inform through the blockchain that the data package is ready and how access to it. The communication process, consent management and data availability are automatised using smart contracts.

Software obfuscation: This thesis includes the study of mechanisms to protect the software's source code for avoiding vulnerability exploitation (i.e. the DAO attack in 2016) and intellectual property protection. This work presents a construction of a control flow graph trans-compilation algorithm that transforms a program into a new functionally equivalent program. This algorithm uses standard instructions (i.e. register and stack operations) and a random routing variable to construct a new control-flow graph. This result can be used in addition to the classic software obfuscation techniques that do not offer protection against getting information using the control-flow graph while the application is running.

Cryptocurrency for new economic models: In this thesis, we proposed the first ERC20 token on a consortium blockchain. This implementation was carried out on top of the Hyperledger Fabric blockchain. The scheme offers the flexibility to create new types of cryptocurrencies or tokens, like the PlasticTokens, allowing communities to create new local or circular economies based on blockchain. More precisely, this work addressed the issue of plastic recycling in Europe. The platform encourages people to participate in reducing their plastic waste by using rewards paid on PlasticTokens. Moreover, PlasticToken can be used to sell or buy products made from recycled plastic through a marketplace implemented on Hyperledger Fabric. The ERC20 token and all the marketplace functionalities are developed in smart contracts, making it the first fully decentralised platform for enabling circular economies.

Open problems

There are several open problems in the blockchain field to address in other to consider this technology as fully ready for cross-industry implementation. The author of this work has selected the open problems presented in this subsection as interesting problems to solve after his thesis.

One interesting open issue is network privacy. Today, there is a lack of results addressing the user privacy vulnerability at the network level, and it is a very sensitive issue to achieve anonymity. We know that unlinkability does not ensure anonymity; hence, adding new mechanisms to hide the source of a transaction could add the extra security layer required to achieve anonymity.

Another open problem is blockchain performance. Today this has been improved considerably in comparison to the first version of Bitcoin. However, this issue is still far to be solved to achieve the same performance levels as a centralised system. The issue related to the performance has different branches; nevertheless, one of the most important ones is the consensus. Achieve performance in a distributed system means to optimise the consensus algorithm. However, the solution is not simple because there is a trade-off between performance and security that is hard to handle in distributed architectures.

Finally, another interesting open problem is tokenomics. This thesis addressed the technical issue to implement new economic models using blockchain technology. Nevertheless, new economic models imply problems much bigger than the implementation of a platform. For example, they cover economic modelling, law, cultural aspects, among others. Hence, implementing a new real circular or local economy model based on blockchain is still an open issue that could be addressed as a multidisciplinary research field.

Bibliography

[ABB$^+$18] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al., *Hyperledger fabric: a distributed operating system for permissioned blockchains*, Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018, ACM, 2018, pp. 30:1–30:15.

[ABF18] Martín Abadi, Bruno Blanchet, and Cédric Fournet, *The applied pi calculus: Mobile values, new names, and secure communication*, J. ACM **65** (2018), no. 1, 1:1–1:41.

[ABFG14] Giuseppe Ateniese, Ilario Bonacina, Antonio Faonio, and Nicola Galesi, *Proofs of space: When space is of the essence*, Security and Cryptography for Networks (Cham) (Michel Abdalla and Roberto De Prisco, eds.), Springer International Publishing, 2014, pp. 538–557.

[ACST06] Man Ho Au, Sherman S. M. Chow, Willy Susilo, and Patrick P. Tsang, *Short linkable ring signatures revisited*, Public Key Infrastructure, Third European PKI Workshop: Theory and Practice, EuroPKI 2006, Turin, Italy, June 19-20, 2006, Proceedings (Andrea S. Atzeni and Antonio Lioy, eds.), Lecture Notes in Computer Science, vol. 4043, Springer, 2006, pp. 101–115.

[ANN06] Michel Abdalla, Chanathip Namprempre, and Gregory Neven, *On the (im) possibility of blind message authentication codes*, Cryptographers Track at the RSA Conference, Springer, 2006, pp. 262–279.

[Att19] Elikem Attah, *Five most prolific 51% attacks in crypto: Verge, ethereum classic, bitcoin gold, feathercoin, vertcoin*, May 2019.

[B$^+$02] Adam Back et al., *Hashcash-a denial of service countermeasure.*

[Ban19] Plastic Bank, *Social plastic*, `https://plasticbank.com/social-plastic-33/`, 2019.

[BBB$^+$07] Elaine B. Barker, William C. Barker, William E. Burr, W. Timothy Polk, and Miles E. Smid, *Sp 800-57. recommendation for key management, part 1: General (revised)*, Tech. report, Gaithersburg, MD, United States, 2007.

[BBB$^+$18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell, *Bulletproofs: Short proofs for confidential transactions and more*, 2018 IEEE Symposium on Security and Privacy (SP), IEEE, 2018, pp. 315–334.

[BF03] Dan Boneh and Matthew K. Franklin, *Identity-based encryption from the weil pairing*, SIAM J. Comput. **32** (2003), no. 3, 586–615.

[BG17] Vitalik Buterin and Virgil Griffith, *Casper the friendly finality gadget*, 2017.

[BHS93a] Dave Bayer, Stuart Haber, and W Scott Stornetta, *Improving the efficiency and reliability of digital time-stamping*, Sequences Ii, Springer, 1993, pp. 329–334.

[BHS93b] Dave Bayer, Stuart Haber, and W. Scott Stornetta, *Improving the efficiency and reliability of digital time-stamping*, Sequences II (New York, NY) (Renato Capocelli, Alfredo De Santis, and Ugo Vaccaro, eds.), Springer New York, 1993, pp. 329–334.

[BKMPK18] Aurelie Bayle, Mirko Koscina, David Manset, and Octavio Perez-Kempner, *When blockchain meets the right to be forgotten: technology versus law in the healthcare industry*, 2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI), IEEE, 2018, pp. 788–792.

[Bon12] Dan Boneh, *Pairing-based cryptography: Past, present, and future*, Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings (Xiaoyun Wang and Kazue Sako, eds.), Lecture Notes in Computer Science, vol. 7658, Springer, 2012, p. 1.

[Bro11] Lyle D Broemeling, *An account of early statistical inference in arab cryptology*, The American Statistician **65** (2011), no. 4, 255–257.

[BSA14] Alysson Bessani, Joao Sousa, and Eduardo EP Alchieri, *State machine replication for the masses with bft-smart*, 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, IEEE, 2014, pp. 355–362.

[CFN88] David Chaum, Amos Fiat, and Moni Naor, *Untraceable electronic cash*, Conference on the Theory and Application of Cryptography, Springer, 1988, pp. 319–327.

[CGJZ01] Stanley Chow, Yuan Gu, Harold Johnson, and Vladimir A. Zakharov, *An approach to the obfuscation of control-flow of sequential computer programs*, International Conference on Information Security, vol. 2200, Springer, 2001, pp. 144–155.

[Cha79] David Lee Chaum, *Computer systems established, maintained and trusted by mutually suspicious groups*, Electronics Research Laboratory, University of California, 1979.

[Cha82] David Chaum, *Blind signatures for untraceable payments*, Advances in Cryptology: Proceedings of CRYPTO '82, Plenum, 1982, pp. 199–203.

[Cha83] David Chaum, *Blind signature system*, Advances in Cryptology, Proceedings of CRYPTO '83, Santa Barbara, California, USA, August 21-24, 1983, 1983, p. 153.

[Cit] Monnaie Locale Complementaire Citoyenne, *Les monnaies locales en france et plus*, `http://monnaie-locale-complementaire-citoyenne.net/france/`, Accessed 19-06-2019 (in French).

[CL99] Miguel Castro and Barbara Liskov, *Practical byzantine fault tolerance*, Proceedings of the Third Symposium on Operating Systems Design and Implementation (USA), OSDI 99, USENIX Association, 1999, p. 173186.

[CNS15] CNSSP, *Policy on the use of commercial solutions to protect national security systems.*

[Coh95] Fred Cohen, *A short history of cryptography*, 1995, [Accessed on 27-06-2021].

[Coi19] CoinMarketCap, *All cryptocurrencies*, `https://coinmarketcap.com/all/views/all/`, 2019.

[Com16] European Commission, *Strategy on plasticsin a circular economy*, `http://ec.europa.eu/smart-regulation/roadmaps/docs/plan_2016_39_plastic_strategy_en.pdf`, 2016, Accessed: 2019-06-11.

[Con16a] MyHealthMyData Consortium, *My health my data.*, `http://www.myhealthmydata.eu/`, 2016, [Online; accessed April. 2020].

[Con16b] PlasticTwist Consortium, *Plastictwist.*, `http://www.ptwist.eu/`, 2016, [Online; accessed April. 2020].

[CP10] Jan Cappaert and Bart Preneel, *A general model for hiding control flow*, Proceedings of the tenth annual ACM workshop on Digital rights management, ACM, 2010, pp. 35–42.

[CYLR18] Marwa Chaieb, Souheib Yousfi, Pascal Lafourcade, and Riadh Robbana, *Verify-your-vote: A verifiable blockchain-based online voting protocol*, Information Systems - 15th European, Mediterranean, and Middle Eastern Conference, EMCIS 2018, Limassol, Cyprus, October 4-5, 2018, Proceedings (Marinos Themistocleous and Paulo Rupino da Cunha, eds.), Lecture Notes in Business Information Processing, vol. 341, Springer, 2018, pp. 16–30.

[Dav15] Lucas Vincenzo Davi, *Code-reuse attacks and defenses*, Ph.D. thesis, 2015.

[Dig] Digiconomist, *Bitcoin energy consumption index*, `https://digiconomist.net/bitcoin-energy-consumption`, Accessed: 2019-06-11.

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer, *Consensus in the presence of partial synchrony*, Journal of the ACM (JACM) **35** (1988), no. 2, 288–323.

[DMPZ14] Sisi Duan, Hein Meling, Sean Peisert, and Haibin Zhang, *Bchain: Byzantine replication with high throughput and embedded reconfiguration*, International Conference on Principles of Distributed Systems, Springer, 2014, pp. 91–106.

[DN93] Cynthia Dwork and Moni Naor, *Pricing via processing or combatting junk mail*, Advances in Cryptology — CRYPTO' 92 (Berlin, Heidelberg) (Ernest F. Brickell, ed.), Springer Berlin Heidelberg, 1993, pp. 139–147.

[DR05] Thomas Dullien and Rolf Rolles, *Graph-based comparison of executable objects (english version)*, SSTIC, vol. 5, 2005, pp. 1–3.

[EARB16] Kimberly Ennis, Stan Aronow, Jim Romano, and Michael Burkett, *The gartner supply chain top 25 for 2016*, `https://www.`

gartner.com/en/documents/3321523, 2016, Accessed: 2019-06-11.

[Eco18] Ecocoin, *Ecocoin*, https://www.ecocoin.com, 2018, Accessed: 2019-06-19.

[Eco19] New Plastics Economy, *New plastics economy global commitment*, https://www.newplasticseconomy.org/assets/doc/GC-Report-June19.pdf, 2019, Accessed: 2019-06-11.

[ET76] Kapali P Eswaran and R Endre Tarjan, *Augmentation problems*, SIAM Journal on Computing **5** (1976), no. 4, 653–665.

[Eur] Plastics Europe, *Plastics europe*, https://www.plasticseurope.com, Accessed: 2019-06-11.

[Fab20] Hyperledger Fabric, *Hypeledger fabricdocs master documentation*, 2020, [Accessed on 17-06-2021].

[FF17] World Economic Forum and Elle McArthur Foundation, *The new plastics economy catalysing action*, http://www3.weforum.org/docs/WEF_NEWPLASTICSECONOMY_2017.pdf, 2017, Accessed: 2019-06-11.

[FFC16] World Economic Forum, Ellen MacArthur Foundation, and McKinsey & Company, *The new plastics economy - rethinking the future of plastics*, https://www.ellenmacarthurfoundation.org/assets/downloads/EllenMacArthurFoundation_TheNewPlasticsEconomy_Pages.pdf, 2016, Accessed: 2019-06-11.

[FHS15] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig, *Practical round-optimal blind signatures in the standard model*, Annual Cryptology Conference, Springer, 2015, pp. 233–253.

[Fis17] Stephanie Shore Fisher, *How to make scratch-off gift cards with just paint and dish soap*, https://www.goodhousekeeping.com/home/craft-ideas/a42597/how-to-make-scratch-off-cards/, 2017, Accessed on 04-07-2019.

[Fla04] Halvar Flake, *Structural comparison of executable objects*, DIMVA 2004, July 6-7, Dortmund, Germany (2004), 161–173.

[Fol12] Followmyvote, *Follow my vote*, https://followmyvote.com/, 2012.

[Fou19] Linux Foundation, *Hyperledger.*, `https://www.hyperledger.org/`, 2019, [Online; accessed April. 2020].

[GHM$^+$17] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich, *Algorand: Scaling byzantine agreements for cryptocurrencies*, Proceedings of the 26th Symposium on Operating Systems Principles, 2017, pp. 51–68.

[GKL15] Juan Garay, Aggelos Kiayias, and Nikos Leonardos, *The bitcoin backbone protocol: Analysis and applications*, Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2015, pp. 281–310.

[Gre15] Gideon Greenspan, *Multichain private blockchain, white paper*, URl: http://www. multichain. com/download/MultiChain-White-Paper. pdf (2015).

[HAM18] Freya Sheer Hardwick, Raja Naeem Akram, and Konstantinos Markantonakis, *E-voting with blockchain: An e-voting protocol with decentralisation and voter privacy*, CoRR **abs/1805.10258** (2018).

[HBG16] Ethan Heilman, Foteini Baldimtsi, and Sharon Goldberg, *Blindly signed contracts: Anonymous on-blockchain and off-blockchain bitcoin transactions*, International Conference on Financial Cryptography and Data Security, Springer, 2016, pp. 43–60.

[HBHW16] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox, *Zcash protocol specification*, Tech. report, Technical report, 2016–1.10. Zerocoin Electric Coin Company, 2016.

[Hes17] Thomas F Heston, *A blockchain solution to gun control*, PeerJ Preprints **5** (2017).

[HM05] Darrel Hankerson and Alfred Menezes, *Elliptic curve cryptography*, Encyclopedia of Cryptography and Security (Henk C. A. van Tilborg, ed.), Springer, 2005.

[HS90] Stuart Haber and W Scott Stornetta, *How to time-stamp a digital document*, Conference on the Theory and Application of Cryptography, Springer, 1990, pp. 437–455.

[HS91] Stuart Haber and W. Scott Stornetta, *How to time-stamp a digital document*, Journal of Cryptology **3** (1991), no. 2, 99–111.

[HWW18] Celine Herweijer, Dominic Waughray, and Sheila Warren, *Building block (chain) s for a better planet.*

[IBM16] IBM, *Tradelens.*, `http://www.tradelens.com/`, 2016, [Online; accessed April. 2020].

[Ibm19] Ibm, *Ibm food trust*, `https://www.ibm.com/uk-en/blockchain/solutions/food-trust`, 2019.

[Inc] BIT Congress Inc., *Bitcongress*, `http://cryptochainuni.com/wp-content/uploads/BitCongress-Whitepaper.pdf`.

[Inc20] Bits Inc, *Tendermint.*, `https://www.tendermint.com/`, 2020, [Online; accessed April. 2020].

[Iro21] Hyperledger Iroha, *Iroha v2.0 - hyperledger iroha documentation*, 2021, [Accessed on 16-06-2021].

[JCJ10] Ari Juels, Dario Catalano, and Markus Jakobsson, *Coercion-resistant electronic elections*, Towards Trustworthy Elections, New Directions in Electronic Voting (David Chaum, Markus Jakobsson, Ronald L. Rivest, Peter Y. A. Ryan, Josh Benaloh, Miroslaw Kutylowski, and Ben Adida, eds.), Lecture Notes in Computer Science, vol. 6000, Springer, 2010, pp. 37–63.

[Kah96] David Kahn, *The codebreakers: The comprehensive history of secret communication from ancient times to the internet*, Simon and Schuster, 1996.

[KKM$^+$05] Christopher Kruegel, Engin Kirda, Darren Mutz, William Robertson, and Giovanni Vigna, *Polymorphic worm detection using structural information of executables*, International Workshop on Recent Advances in Intrusion Detection (RAID'05), vol. 3858, Springer, 2005, pp. 207–226.

[KL20] Jonathan Katz and Yehuda Lindell, *Introduction to modern cryptography*, CRC press, 2020.

[KLMN18] Mirko Koscina, Pascal Lafourcade, David Manset, and David Naccache, *Blindcons: A consensus algorithm for privacy preserving private blockchains*, Tech. report, LIMOS, 2018, `http://sancy.univ-bpclermont.fr/~lafourcade/BlindCons.pdf`.

[KN12] Sunny King and Scott Nadal, *Ppcoin: Peer-to-peer crypto-currency with proof-of-stake*, self-published paper, August **19** (2012).

[KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov, *Ouroboros: A provably secure proof-of-stake blockchain protocol*, Advances in Cryptology – CRYPTO 2017 (Cham) (Jonathan Katz and Hovav Shacham, eds.), Springer International Publishing, 2017, pp. 357–388.

[LD03] Cullen Linn and Saumya Debray, *Obfuscation of executable code to improve resistance to static disassembly*, Proceedings of the 10th ACM conference on Computer and communications security, ACM, 2003, pp. 290–299.

[Ler15] Xavier Leroy, *The compcert c verified compiler: Documentation and users manual*, Ph.D. thesis, Inria, 2015.

[LK09] Tımea László and Ákos Kiss, *Obfuscating C++ programs via control flow flattening*, Annales Universitatis Scientarum Budapestinensis de Rolando Eötvös Nominatae, Sectio Computatorica **30** (2009), 3–19.

[LM07] Jinyuan Li and David Maziéres, *Beyond one-third faulty replicas in byzantine fault tolerant systems*, Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation (USA), NSDI07, USENIX Association, 2007, p. 10.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease, *The byzantine generals problem*, ACM Transactions on Programming Languages and Systems (TOPLAS) **4** (1982), no. 3, 382–401.

[Ltd06] Cypher Research Laboratories Pty. Ltd., *History of cryptography*, 2006, [Accessed on 27-06-2021].

[Med19] Medicalchain, *Medicalchain - blockchain for electronic health records*, `https://medicalchain.com/en/`, 2019.

[Mer79] Ralph Charles Merkle, *Secrecy, authentication, and public key systems.*, Stanford university, 1979.

[Mon15] John V. Monaco, *Identifying Bitcoin users by transaction behavior*, Biometric and Surveillance Technology for Human and Activity Identification XII (Ioannis A. Kakadiaris, Ajay Kumar, and Walter J. Scheirer, eds.), vol. 9457, International Society for Optics and Photonics, SPIE, 2015, pp. 25 – 39.

[MSH17] Patrick McCorry, Siamak F. Shahandashti, and Feng Hao, *A smart contract for boardroom voting with maximum voter privacy*, FC 2017, vol. 10322, Springer, 2017.

[Nak08] Satoshi Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, November 2008.

[NT15] National Institute of Standards NIST and Technology, *National institute of standards and technology nist (no date) glossary - data governance*, `https://csrc.nist.gov/glossary/term/data_governance`, 2015, Accessed on 20-09-2020.

[OBL16] OBL, *Ibm (no date) data governance*, `https://www.ibm.com/analytics/data-governance`, 2016, Accessed on 20-09-2020.

[Oka92] Tatsuaki Okamoto, *Provably secure and practical identification schemes and corresponding signature schemes*, Annual International Cryptology Conference, Springer, 1992, pp. 31–53.

[Oka06] Okamoto, Tatsuaki, *Efficient blind and partially blind signatures without random oracles*, 2006.

[oSN15] National Institute of Standards and Technology NIST, *Hyperledger (2019). introduction.*, `https://hyperledger-fabric.readthedocs.io/en/release-1.4/blockchain.html`, 2015, Accessed on 20-09-2020.

[P4T14] P4Titan, *Slimcoin a peer-to-peer crypto-currency with proof-of-burn*, Tech. report, 2014.

[Pai11] Pascal Paillier, *Paillier encryption and signature schemes*, Encyclopedia of Cryptography and Security, 2nd Ed. (Henk C. A. van Tilborg and Sushil Jajodia, eds.), Springer, 2011, pp. 902–903.

[Par16] European Parlament, *Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation)*, `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:02016R0679-20160504`, 2016, Accessed on 20-09-2020.

[PDA07] Igor V. Popov, Saumya K. Debray, and Gregory R. Andrews, *Binary obfuscation using signals*, Usenix Security, 2007.

[PK15] Jack Peterson and Joseph Krug, *Augur: a decentralized, open-source platform for prediction markets*, CoRR **abs/1501.01042** (2015).

[Pon18] Institute Ponemon, *cost of a data breach study: Global overview*, Benchmark research sponsored by IBM Security Independently conducted by Ponemon Institute LLC (2018).

[PSL80] Marshall Pease, Robert Shostak, and Leslie Lamport, *Reaching agreement in the presence of faults*, Journal of the ACM (JACM) **27** (1980), no. 2, 228–234.

[PUT07] S. Narayan P. Udaya and V. Teague, *A secure electronic voting scheme using identity based public key cryptography*, Proceedings of SAR-SSI 2007, Annecy, France, June 12-16, 2007, 2007.

[Quo16] Quorum, *Quorum whitepaper*, `https://github.com/jpmorganchase/quorum/blob/master/docs/QuorumWhitepaperv0.2.pdf`, 2016.

[Rag05] S. Raghavan, *A note on* E*swaran and* T*arjans algorithm for the strong connectivity augmentation problem*, The Next Wave in Computing, Optimization, and Decision Technologies, vol. 29, Springer, 2005, pp. 19–26.

[reg19] Global Property register, *Global property register on the blockchain*, `https://basicattentiontoken.org/BasicAttentionTokenWhitePaper-4.pdf`, 2019.

[Ric53] Henry Gordon Rice, *Classes of recursively enumerable sets and their decision problems*, Transactions of the American Mathematical Society **74** (1953), no. 2, 358–366.

[RS04] Phillip Rogaway and Thomas Shrimpton, *Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance*, Fast Software Encryption (Berlin, Heidelberg) (Bimal Roy and Willi Meier, eds.), Springer Berlin Heidelberg, 2004, pp. 371–388.

[RS15] Francesco Rossi and Giovanni Schmid, *Identity-based secure group communications using pairings*, Computer Networks **89** (2015), 32–43.

[Sch91] Claus-Peter Schnorr, *Efficient signature generation by smart cards*, Journal of cryptology **4** (1991), no. 3, 161–174.

[Sch01] Claus Peter Schnorr, *Security of blind discrete log signatures against interactive attacks*, International Conference on Information and Communications Security, Springer, 2001, pp. 1–12.

[Sci20] Coin Sciences, *Multichain.*, `https://www.multichain.com/`, 2020, [Online; accessed April. 2020].

[SK11] Sebastian Schrittwieser and Stefan Katzenbeisser, *Code obfuscation against static and dynamic reverse engineering*, International Workshop on Information Hiding, Springer, 2011, pp. 270–284.

[SKK$^+$16] Sebastian Schrittwieser, Stefan Katzenbeisser, Johannes Kinder, Georg Merzdovnik, and Edgar Weippl, *Protecting software through obfuscation: Can it keep pace with progress in code analysis?*, ACM Computing Surveys (CSUR) **49** (2016), no. 1, 4.

[Sma16] Smartmatic, *Tivi*, `http://www.smartmatic.com/voting/online-voting-tivi/`, 2016.

[SMD14] Amitabh Saxena, Janardan Misra, and Aritra Dhar, *Increasing anonymity in bitcoin*, International Conference on Financial Cryptography and Data Security, Springer, 2014, pp. 122–139.

[Sof18] Brave Software, *Basic attention token (bat)*, `https://basicattentiontoken.org/BasicAttentionTokenWhitePaper-4.pdf`, 2018.

[Sol18] Solidity, *Introduction to smart contracts - solidity 0.4.24 documentation*, 2018, [Accessed on 17-06-2021].

[Ste12] Iain Stewart, *Proof of burn - bitcoin wiki*, 2012, [Accessed on 16-06-2021].

[Swa15] Tim Swanson, *Consensus-as-a-service: a brief report on the emergence of permissioned, distributed ledger systems*, Report, available online, Apr (2015).

[Sza96] Nick Szabo, *Smart contracts: building blocks for digital markets*, EXTROPY: The Journal of Transhumanist Thought,(16) **18** (1996), no. 2.

[TL18] Ken Timsit Tom Lyons, Ludovic Courcelas, *Blockchain and gdpr*, Tech. report, The european union blockchain observatory and forum, oct 2018.

[VB15] Fabian Vogelsteller and Vitalik Buterin, *Erc 20*, `https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md`, 2015, Accessed on 02-07-2019.

[VS13] Nicolas Van Saberhagen, *Cryptonote v 2.0*, 2013.

[Wat17] Roger Peter Wattenhofer, *Distributed ledger technology: The science of the blockchain*, Inverted Forest Publishing, 2017.

[WHKD00] Chenxi Wang, Jonathan Hill, John Knight, and Jack Davidson, *Software tamper resistance: Obstructing static analysis of programs*, Tech. report, Technical Report CS-2000-12, University of Virginia, 12 2000, 2000.

[Woo14] Gavin Wood, *Ethereum: A secure decentralised generalised transaction ledger*, Ethereum Project Yellow Paper **151** (2014), 1–32.

[YLS$^+$18] Bin Yu, Joseph K. Liu, Amin Sakzad, Surya Nepal, Ron Steinfeld, Paul Rimba, and Man Ho Au, *Platform-independent secure blockchain-based voting system*, Information Security - 21st International Conference, ISC 2018, Guildford, UK, September 9-12, 2018, Proceedings (Liqun Chen, Mark Manulis, and Steve Schneider, eds.), Lecture Notes in Computer Science, vol. 11060, Springer, 2018, pp. 369–386.

RÉSUMÉ

Cette thèse aborde les problèmes de sécurité, de confidentialité et des barrières économiques auxquels fait face la mise en œuvre de la blockchain pour assurer la confidentialité des utilisateurs, la gouvernance des données, l'offuscation des logiciels et développer des modèles de crypto-monnaies pour les économies circulaires. Après avoir introduit la blockchain et la cryptographie moderne, sont présentés trois principaux résultats de recherche : un nouveau protocole de consensus qui préserver la confidentialité des utilisateurs, un nouveau schéma de vote électronique confidentiel utilisant une blockchain permissioned et un protocole d'offuscation de graphiques de flux de contrôle pour la confidentialité des logiciels. Enfin, la troisième partie de la thèse comprend deux résultats de recherche appliquée avec deux prototypes de systèmes blockchain répondant aux problématiques posées par la gouvernance des données et les crypto-monnaies.

La première partie de cette thèse décrit un bref historique de la technologie autour de la blockchain et son évolution jusqu'à arriver à sa première implémentation (le Bitcoin). Ensuite, sont introduits les principaux sujets sur la cryptographie moderne utilisés dans cette thèse. La deuxième partie commence par un nouvel algorithme de consensus qui préserve la confidentialité pour la blockchain *permissioned*. La principale contribution de ce travail est une construction générale de signature aveugle pour la dissociation des transactions intégrée à un consensus basé sur le BFT pour la validation des transactions et des blocs. Le deuxième résultat correspond à un nouveau schéma de vote électronique pour la blockchain *permissioned*. Ce travail a propose un nouveau consensus de préservation de la confidentialité basé sur la signature aveugle d'Okamoto-Schnorr pour la dissociation des transactions et la courbe elliptique pour la confidentialité des votes. Enfin, il est présenté un nouveau mécanisme d'offuscation de graphes de flux de contrôle pour la confidentialité du source code des logiciels (par exemple, pour les contrats intelligents).La troisième partie présente deux implémentations de la blockchain. Le premier est une nouvelle implémentation de la norme de jeton ERC20 et une marketplace sur Hyperledger Fabric pour les économies circulaires supportées par le recyclage du plastique. Le second est un prototype de blockchain pour la gouvernance des données dans l'échange de données de santé selon le RGPD automatisé avec des contrats intelligents.

MOTS CLÉS

blockchain, distributed systemes, permissioned blockchain, blind signature, eVoting, Tokenomics, GDPR.

ABSTRACT

This thesis addresses the security, privacy, and economic barriers to implementing blockchain for user's privacy, data governance, software obfuscation and cryptocurrencies models for circular economies. The first part of this thesis covers the preliminaries about blockchain and modern cryptography. The second part presents three main results concerning privacy-preserving consensus, privacy in electronic voting with permissioned blockchain, and control-flow graph obfuscation for software confidentiality. Finally, the third part includes two results based on applied research work with two prototypes of blockchain systems for data governance and cryptocurrencies.
The first part of this thesis describes a brief history of the technology around blockchain and its evolution until arriving at the first blockchain implementation (Bitcoin). Then, the most relevant topics on Modern Cryptography used in this thesis are introduced.
The second part begins with a new privacy-preserving consensus algorithm for permissioned blockchain. The main contribution of this work is a general construction of blind signature for transaction unlinkability integrated to a BFT-based consensus for transactions and blocks validation. The second result corresponds to a new electronic voting scheme for permissioned blockchain. This work proposed a new privacy-preserving consensus based on Okamoto-Schnorr blind signature for transaction unlinkability and elliptic curve for vote privacy. Finally, a new control-flow graph obfuscation mechanism for software confidentially (e.i for smart contracts) is presented.
The third part presents two blockchain implementations. The first is a novel implementation of the ERC20 token standard and a Marketplace on Hyperledger Fabric for circular economies based on plastic recycling. The second one is a blockchain prototype for data governance in healthcare data exchange according to GDPR automatised using smart contracts.

KEYWORDS

blockchain, distributed systemes, permissioned blockchain, blind signature, eVoting, Tokenomics, GDPR.